jamk.fi

Combitech Oy:n ohjelmistotuotannon prosessikuvaus

Jarmo Luostarinen

Opinnäytetyö
Huhtikuu 2019
Tekniikan ja liikenteen ala
Insinööri (AMK), Tieto- ja viestintätekniikka
Ohjelmistotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Luostarinen, Jarmo	Opinnäytetyö, AMK	Huhtikuu 2019
	Sivumäärä 62	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Combitech Oy:n ohjelmistotuotannon prosessikuvaus

Tutkinto-ohjelma
Tieto- ja viestintätekniikan tutkinto-ohjelma, Ohjelmistotekniikka

Työn ohjaaja(t)
Marko Rintamäki, Jouni Huotari

Toimeksiantaja(t)
Combitech Oy

Tiivistelmä

Opinnäytetyön aihe valittiin toimeksiantajan tarpeesta yhtenäistää eri projektien tuotantoprosesseja ja näin tehostaa yrityksen henkilöresurssien käyttöä.

Combitech Oy tuottaa räätälöityjä ohjelmistoja asiakkailleen ja ylläpitää toimitettuja ohjelmistoja. Tavoitteena oli dokumentoida Combitech Oy:n ohjelmistotuotannon prosessikuvaus ja kehittää toimipisteestä riippumatonta yhtenäistä tuotanto prosessia. Työn pohjana oli laaja perehtyminen ketteriin ohjelmistokehitysmalleihin useita erilaisia lähteitä käyttäen.

Toinen tärkeä tavoite oli luoda prosessikuvauksesta dokumentti uusien työntekijöiden perehdytystä varten. Henkilöresurssien käyttö yli eri projektien on Combitech Oy:n tyyppisessä yrityksessä lähes välttämätöntä, joten yhtenäinen tuotantoprosessi projekteissa helpottaa resurssien jakoa, sekä lyhentää uusien työntekijöiden perehdytykseen menevää aikaa.

Tuotantoprosessin kehitystä varten pidettiin säännöllisesti palavereita, joihin osallistui Combitech Oy:n henkilökuntaa teknologiajohtajasta ohjelmistokehittäjiin. Näin saatiin kehitys koskemaan koko tuotantoprosessia.

Työ toteutettiin Combitech Oy:lle ja työn lopputuloksena oli skaalautuvaa tuotannon prosessia kuvaava dokumentti, joka ottaa kantaa ylemmän ja alemman tason työn kuvaukseen ja toimintatapoihin, tiimitason työn kuvaukseen ja toimintatapoihin, sekä lähdekoodin versionhallintaan, jonka kuvaus oli yksi työn tärkeä osa.

Avainsanat (asiasanat)
Tuotantoprosessi, Ketterä kehitys, Lähdekoodin versionhallinta, Git, Ohjelmistokehitys

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Luostarinen, Jarmo	Bachelor's thesis	April 2019
		Language of publication: Finnish
	Number of pages 62	Permission for web publication: x

Title of publication

Process description of the software production of Combitech Oy

Degree programme

Information and Communications Technology, Software Engineering

Supervisor(s)

Rintamäki Marko, Huotari Jouni

Assigned by

Combitech Oy

Abstract

The subject of the thesis was chosen from the assignor's need to standardize the production process for different projects so that the human resources could be shared more easily over the different projects. Combitech Oy is a software company that produces tailored software products and maintains those products.

One of the main purposes was to document the process description of Combitech Oy's software production process so that the process could be used as a unit-independent process. That way the documentation of the software production process could be a part of the Combitech Oy's orientation material for their new employees, which was the second main objective of the research.

The work was based on studying different kind of agile software development methods from a variety of relevant sources.

Meetings were held repeatedly for developing the software production process, and employees from the Director of Technology to software developers participated in these meetings so that the development of the process would affect all parts of the development process.

The third objective of the study were the methods of source code version management and standards description.

The outcome of the project was a document of a scalable agile software production process that takes into account high-level work methods and visualization as well as team-level work methods and visualization.

Keywords/tags (subjects)

Production Process, Agile, Source Code Version Control, Git, Software Development

Miscellaneous (Confidential information)

Sisältö

Kuviot

Sanasto

Agile Release Train (ART):

Agile Release Train on viidestä kahteentoista kehitystiimistä ja tukitiimistä koostuva ryhmä, jonka tavoitteet suunnitellaan ja määritellään SAFessa Program Increment suunnittelussa.

Backlog:

Ohjelmistokehityksen työlista, joka sisältää kehitettävän ohjelmiston toteutusta odottavat kehitysehdotukset ja parannusehdotukset.

Definition of done (DoD):

Valmiin määritelmä kertoo vaiheittain, mitkä seikat tulee ottaa huomioon ennen kuin kunkin vaiheen voidaan sanoa olevan valmis.

Delivery:

Toimitus, eli esimerkiksi pienen vikakorjauksen jälkeen asiakkaalle toimitettava versio ohjelmistosta tai kehityksen aikana testaukseen toimitettava versio ohjelmistosta, joka ei ole käynyt julkaisuun liittyvää prosessia läpi.

DevOps:

Ketterä toimintamalli kehitysympäristön ja tuotannon tehokkaaseen yhdistämiseen, jonka tarkoitus on mahdollistaa nopeat toimitukset tuotantoon.

Epic:

Laaja ohjelmiston toiminnallisuus, jota käsitellään Program-tason työn vaiheiden kuvauksessa. Epic eli toiminnallisuus pilkotaan käyttäjätarinoiksi tiimin kehitettäväksi.

Git:

Lähdekoodin hajautettu versionhallinta, jolla varmistetaan versiointi ja lähdekoodin eheys kehityksen aikana.

Kanban:

Ketterän kehityksen toteutusmalli, joka perustuu jatkuvaan toteutukseen ja kerrallaan työn alla olevien työmäärien rajoittamiseen.

Ketterä kehitys:

Ohjelmistokehitystä, joka on nopeasti muutoksiin reagoivaa, kommunikaatioon perustuvaa ja nopeisiin toimituksiin pyrkivää.

Lean-ajattelumalli:

Materiaalin ja työtuntien hukan minimoimiseen pyrkivä kehityksen ajattelumalli, joka on alun perin otettu käyttöön Toyotan autotehtaalla.

Program-taso:

Program-tasolla tarkoitetaan ohjelmiston laajempien kokonaisuuksien kuvausta, eli esimerkiksi kehityksen seuranta Program-tasolla keskittyy Epic-tyyppisten tehtävien seurantaan, eikä päivittäiseen kehitystyön seurantaan.

Regressiotestaus:

Aiemmin tuotettujen ja ajettujen testien uudelleen ajoa ohjelmistolle, kun ohjelmakoodia on muutettu. Tällä varmistetaan ohjelmiston eheys muutosten jälkeen.

Release:

Release eli julkaisu on asiakkaalle toimitettava, testattu ja ennalta suunniteltu ohjelmistokehityksen versio. Julkaisu käy aina läpi julkaisuprosessin ennen asiakkaalle toimittamista.

Scaled Agile Framework (SAFe):

Scaled Agile Framework on skaalautuva ketterän kehityksen viitekehys, joka ottaa kantaa koko tuotantoprosessiin. Se määrittelee erilaiset prosessin osat ja antaa kattavan kuvauksen tuotannon prosessista ottamatta kantaa yksityiskohtaisiin toimintatapoihin.

Scrum:

Ketterän kehityksen toteutusmalli, joka perustuu 2-4 viikkoon kestäviin kehityssykleihin, joihin ohjelmistokehitys on jaettu.

Scrumban:

Scrumin ja Kanbanin hybridimalli, jossa yhdistetään kahden toteutusmallin hyväksi havaittuja puolia. Alun perin suunniteltu käytettäväksi Scrumista Kanbaniin siirtymistä helpottamaan.

Scrum Master:

Scrum-ohjelmistokehitysmallia käyttävän tiimin tiiminvetäjä, joka hoitaa pääasiassa kommunikoinnin tiimin ulkopuolelle kehityssyklien aikana ja valvoo Scrumin periaatteiden noudattamista kehityksessä.

Sprint:

Ohjelmistokehityksen kehityssykli, joka pituus on 2-4 viikkoa.

Sub-Task / Task:

Jiran käyttäjätarinaa pienempi tehtävän kuvaus, joka usein liittyy työn alle otettuun käyttäjätarinaan.

Systeemi:

Laaja ohjelmistokokonaisuus

Team-taso:

Team-taso eli tiimitaso on tarkoitettu tiimin päivittäisellä tasolla tapahtuvan toiminnan kuvaamiseen

Tuotantoympäristö:

Tuotantoympäristöllä tarkoitetaan yleisesti asiakkaan ympäristöä, jossa ohjelmisto tullaan ottamaan käyttöön.

User Story:

Lausetasolla kuvattu ohjelmiston käyttäjätarina, jonka pohjalta ohjelmakoodi toteutetaan.

Validointi:

Toiminnan tarkoituksenmukaisuuden todentamista, eli varmistetaan, että ohjelmiston toteutus vastaa asiakkaan odotuksia.

Verifiointi:

Vaatimustenmukaisuuden todentamista, eli ohjelmisto toimii, kuten on suunniteltu.

1 Työn lähtökohdat

Combitechin tuotantoprosesseissa on siirrytty viimeisen viiden vuoden aikana ketterästä vesiputousmallista nykyiseen malliin, joka edustaa ketterää ohjelmistokehitystä.

Viimeisten vuosien aikana on tunnistettu, että tuotannon käytänteet vaativat tueksi yhtenäisen prosessimallin, jotta tuotantoa voidaan kehittää vastaamaan nykyisiä ja tulevia tarpeita.

Prosessin kuvauksessa ei ole tarkoitus ottaa kantaa prosessien standardien mukaisuuteen, vaan keskittyä kuvaamaan prosessissa käytettävät menetelmät. Lisäksi ohjelmistotuotantoon käytettäviä työkaluja on todella paljon, mutta tässä opinnäytetyössä otettiin tarkasteltavaksi vain muutamia laajalti tunnettuja työkaluja.

1.1 Tausta

Ennen opinnäytetyön aloitusta tilaajalla oli jo ketterät kehitysmenetelmät käytössään projekteissa. Yrityksessä käytettiin Scrumia ja Kanbania vaihtelevasti eri projekteissa. Tuotantoprosesseissa eri toimipisteillä oli myös joitain eroavaisuuksia.

Tilaaja oli tunnistanut yrityksen tuotantoprosessien yhdenmukaistamisessa paljon etuja, joten yhtenäistä tuotannon prosessikuvausta ryhdyttiin määrittelemään.

Toimialan muutokset sekä asiakkaan hankintoihin liittyvät muutokset edellyttävät alalla toimivilta yrityksiltä tuotantoprosessien kehitystä ja vakioimista. Kiinteähintaiset ennalta suunnitellut projektit ovat haastavia ja alkuvaiheen suunnitelmat harvoin pysyvät muuttumattomina projektin loppuun saakka. Tämä lisää haasteita ja ennalta odottamatonta suunnittelutyötä. On siis tunnistettu tarve siirtyä lähemmäksi ”jatkuvaa” suunnittelua. Tuotannosta on tullut koko ajan ketterämpää ympäristön asettamien rajoitusten puitteissa.

Uusi tuotannonprosessikuvaus painottaa entistä enemmän suunnittelun tärkeyttä ja testauksen merkitystä jo prosessin alkuvaiheessa osana kehitystä. Samalla uusi prosessi mahdollistaa tuotannon seuraamisen korkeammalla tasolla visualisoiden laajojen toiminnallisuuksien kehitystä.

Tuotannon prosessin perustana toimii Essential Scaled Agile Framework (SAFe), mutta malli on käytössä vain soveltuvilta osin, koska SAFe on suunniteltu yli 50 henkilön projekteille. Combitech Oy:n projektit ovat pääsääntöisesti toteutettu pienemmällä henkilöstömäärällä.

1.2 Toimeksiantaja

Työn toimeksiantajana toimi Combitech Oy, joka on puolustusteollisuudessa toimiva räätälöityjen ohjelmistojen kehitysyritys. Yrityksessä työskentelee tällä hetkellä noin 80 henkilöä. Combitech Oy on osa pohjoismaista Combitech yritystä, jossa työskentelee yhteensä noin 1900 henkilöä Suomessa, Ruotsissa, Norjassa ja Tanskassa.

1.3 Tehtävät ja tavoitteet

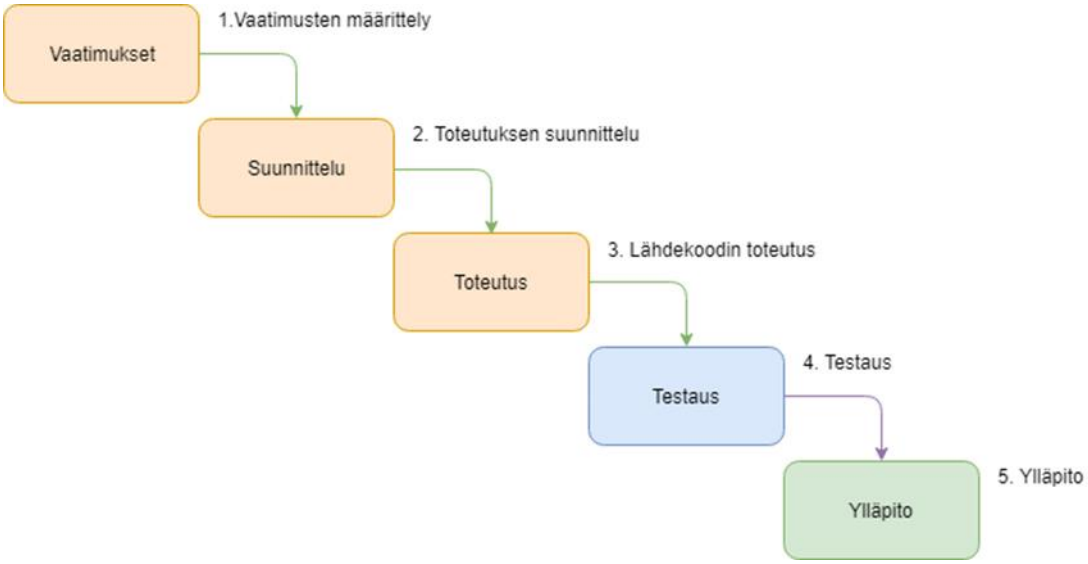
Yrityksessä oli otettu käyttöön Atlassian tuoteperheen työkalut ja näiden työkalujen yhdenmukaisten käytänteiden kuvaaminen oli yksi tärkeimmistä tavoitteista, joita työllä oli.

Toinen tärkeä tavoite oli luoda yhtenäiset lähdekoodin versionhallinnan käytänteet, jolloin kehittäjien ei tarvitse opetella erilaisia versionhallinnan käytänteitä siirtyessä projektilta toiselle.

Kolmas tavoite oli luoda yhtenäinen prosessinkuvaus, joka toimisi uusien työntekijöiden perehdytysmateriaalina.

2 Ohjelmistotuotanto

Ohjelmistotuotanto on jaettu karkeasti viiteen eri osioon, jotka ovat määrittely, suunnittelu, toteutus, testaus ja laadun varmistus. (Haikala & Mikkonen 2011, 12.) Kuviossa 1. on kuvattu perinteinen vesiputousmalli vaiheittain. Kappaleessa 2.1-2.4 on avattu ohjelmistotuotannon vaiheita kattavammin.



Kuvio 1. Vesiputousmalli (Lähteinen 2015, 11.)

1. Vaatimusten määrittely:
   Ensimmäinen vesiputousmallin vaihe on vaatimusmäärittely, jossa ohjelmiston vaatimukset tunnistetaan ja kirjataan ylös. Vaihe on valmis, kun kaikki tunnistettavat vaatimukset on määritelty.

2. Toteutuksen suunnittelu:
   Suunnitteluvaiheessa suunnitellaan arkkitehtuuri ja ohjelmiston kehityksessä olevan osan rakenne lopulliseen muotoonsa, jonka jälkeen vaihe päätetään ja näiden suunnitelmien mukaan mennään seuraavaan vaiheeseen palaamatta enää takaisin suunnitteluun.

3. Lähdekoodin toteutus:
   Tämä vaihe vesiputousmallissa on lähdekoodin toteutusta varten. Tässä kohtaa tulee itse lähdekoodi valmiiksi, jonka jälkeen vaiheeseen ei enää palata.

4. Testaus:
   Tässä kohtaa vesiputousmallia toteutus testataan ja lähdekoodista löydetyt virheet korjataan. Tämän vaiheen lopussa toteutus on valmis osaksi ohjelmistoa.
   Yhdestä vesiputouksen kierroksesta syntyy versio ohjelmistosta. Toisinaan prosessi on tarvetta toistaa useammin, jotta ohjelmisto vastaa sille asetettuja odotuksia ja vaatimuksia.

5. Ylläpito:
   Nimensä mukaisesti ohjelmisto on tässä vaiheessa siirtynyt ylläpitovaiheeseen joka kestää niin kauan, kunnes ohjelmiston tuki lopetetaan.

2.1 Määrittelyvaihe

Määrittely tapahtuu joko vesiputousmallin mukaisesti prosessin alkuvaiheessa tai ketteriä menetelmiä, kuten scrumia käytettäessä useita kertoja läpi toteutuksen.

Määrittelyn aikana on tarkoitus kuvata ohjelmiston toimintaa yleisellä tasolla ja kuvata tuotteen odotuksia ja reunaehtoja, jotka täyttävät tilaajan tarpeet. Nämä toiminnan yleiset kuvaukset ja reunaehdot mahdollistavat teknisen tason tarkemman suunnittelun, joka on kuvattu luvussa 2.2. (Haikala & Mikkonen 2011, 69-70.)

Määrittelyn on tarkoitus olla mahdollisimman selkokielistä ja ymmärrettävää niin tilaajan, kuin toimittajankin kannalta. Määrittely tehdään usein selkokielisenä tekstinä kirjoitettuna ja tarvittaessa apuna käytetään kuvia, tai kaavioita selkeyttämään määrittelyä. Yksi hyvä tapa kuvata ohjelmiston rakennetta on käyttää UML-kaavioita, joita on käytetty ohjelmistotekniikassa jo pitkään. (Haikala & Mikkonen 2011, 69-76.)

Määrittelyvaiheesta saadaan ohjelmiston vaatimukset, jotka tulevat liiketoiminnallisista sekä käyttäjätarpeista. Vaatimusten määrittelyä voidaan toteuttaa monin eri keinoin, kuten käsitemallien, käyttäjätutkimuksien ja haastattelujen avulla.

2.1.1 Ohjelmiston liiketoiminnalliset tarpeet

Liiketoiminnalliset tarpeet ovat yleensä asiakkaan näkökulmasta mietittyjä taloudellista hyötyä tuottavia vaatimuksia, jotka voidaan saavuttaa joko kustannuksia vähentäen, tai vaihtoehtoisesti tuottoa lisäämällä. (Aaronen 2017, 10.)

Yksi hyvä keino kartoittaa liiketoiminnallisia tarpeita on kilpailijavertailut, joissa kartoitetaan kilpailijoiden vahvuuksia ja puutteita. Vertailun pohjalta saadaan priorisoitua omia vaatimuksia ja toiminnallisuuksia. (Aaronen 2017, 10.)

Toinen hyvä tapa liiketoiminnallisten tarpeiden kartoitukseen on tarveanalyysihaastattelut, joilla voidaan selvittää liiketoiminnallisia tarpeita kaikilta ohjelmiston käyttöön liittyviltä sidosryhmiltä, kuten käyttäjiltä, tai myynti- ja markkinointihenkilökunnalta. (Aaronen 2017, 10.)

2.1.2 Ohjelmiston käyttäjien tarpeet

Käyttäjien tarpeiden kartoitus alkaa usein käyttäjätutkimuksella, jolla pyritään määrittelemään erilliset ohjelmistoon liittyvät sidosryhmät ja heidän erilaiset tarpeensa. Käyttäjätutkimuksia on hyvin erilaisia muutaman tunnin haastatteluista aina laajoihin viikkoja kestäviin tutkimuksiin riippuen ohjelmiston laajuudesta ja käytössä olevasta budjetista. (Aaronen 2017, 10.)

Käyttäjätutkimuksen jälkeen tehdään samankaltaisuusanalyysi, jossa kartoitetaan käyttäytymismalleja, yleisiä ongelmia, käyttäjien haluja ja tarpeita, ja sitä, ketkä käyttäjistä luovat eniten hyötyä asiakkaalle ja toisaalta kenen palveleminen tuottaa vähiten taloudellista hyötyä asiakkaalle. (Aaronen 2017, 11.)

Määrittelyssä hyödynnettävät persoonat pohjautuvat käyttäjätutkimuksessa tunnistettuihin käyttäjäryhmiin. Näitä persoonia hyödynnetään, määrittelyssä esimerkiksi roolileikein, jotka helpottavat ymmärtämään erilaisia ohjelmiston käyttötapoja. Näihin persoonien kuvauksiin pyritään keräämään vain kyseisen persoonan muista käyttäjistä yksilöivä tieto, jotta persoona ei paisu liian suureksi, mutta kuitenkin yksilöi persoonan selkeästi muista käyttäjistä. (Aaronen 2017, 11.)

2.1.3 Toiminnallisten ja ei-toiminnallisten vaatimusten määrittely

Ohjelmiston vaatimukset jaetaan kahteen eri ryhmään, jotka ovat toiminnalliset ja ei-toiminnalliset vaatimukset. Toiminnalliset vaatimukset ovat toimintaa kuvaavia vaatimuksia ja ei-toiminnalliset taas ohjelmiston teknisiin määräyksiin, rajoituksiin, ohjeistukseen ja toimintaympäristöön liittyviä vaatimuksia. (Aaronen 2017, 11-12.)

Vaatimuksien määrittelyyn käytetään Toimintatarinoita, jotka pyritään luomaan yhden persoonan, tai käyttäjän näkökulmasta ja kuvaamaan nykyistä ohjelmiston toimintaa konkreettisesti. Näiden kuvausten perusteella taas luodaan Käyttötarinat, jotka kuvaavat ohjelmiston tavoitetilan toiminnallisuutta yhden persoonan näkökulmasta. Yksi oleellinen määrittelyn kuvaus on Käyttäjätarina, joka kuvaa yhden ohjelmiston osa-alueen toimintaa yhden käyttäjän näkökulmasta. Nämä käyttäjätarinat kuvaavat ohjelmiston käyttäjävaatimuksia ja näitä tarinoita voi olla ohjelmiston laajuudesta riippuen todella paljon, jolloin käyttäjätarinoita täytyy priorisoida. Ohjelman toiminnallisia vaatimuksia määritellään paljon käyttötapausten avulla, jotka kuvaavat käyttäjän ja ohjelmiston toiminnon vuorovaikutusta yleisellä tasolla. Nämä käyttötapaukset voivat sisältää useamman käyttäjän käyttötarinoita, joten sanallisten kuvauksien lisäksi voidaan käyttää UML-kielellä toteutettua käyttötapauskaaviota, jossa käyttäjät yhdistetään käyttötapauksiin. Tämän kaavion tarkoitus on hahmottaa kaikki ohjelmistolla olevat toimintatavat, sekä käyttäjävaatimukset, jotka ohjelmistolla on olemassa. (Aaronen 2017, 12-13.)

Näitä tarinoita, tapauksia ja tarpeita kuvatessa määritellään ohjelmiston vaatimukset, jotka priorisoidaan niiden tuottaman lisäarvon ja kustannusten mukaan. Tästä syntyy ohjelmiston vaatimusdokumentti, joka voi olla esimerkiksi Excel-taulukko. Vaatimusdokumentti katselmoidaan läpi projektin sidosryhmien kanssa, jonka jälkeen ohjelmiston omistaja hyväksyy dokumentin, tai vaatii tähän tarvittavat muutokset, jonka jälkeen vaatimusdokumentti voidaan hyväksyä. (Aaronen 2017, 13.)

2.2 Suunnitteluvaihe

Suunnittelun tarkoitus on tuottaa asiakkaalta saatujen vaatimusten pohjalta tekninen kuvaus siitä, miten toteutus tulee täyttämään asetetut vaatimukset. (Haikala & Mikkonen 2011, 177.)

2.2.1 Arkkitehtuurisuunnittelu

Arkkitehtuurisuunnittelu on keskeisin suunnittelun osa-alue, jolla toteutus jaetaan erillisiin komponentteihin ja määritellään toteutuksen rajapinnat. Arkkitehtuurisuunnittelu määrittelee ohjelmiston hierarkian ja keskeisen terminologian. Jotta käsitteistön avulla olisi helppo kommunikoida, täytyy arkkitehtuurisuunnitelman olla selkeä ja suoraviivainen. (Haikala & Mikkonen 2011, 178-180.)

2.2.2 Toiminnallisuuden suunnittelu

Toiminnallisuuden suunnitteluun voidaan käyttää erilaisia mallinnukseen tarkoitettuja ohjelmia, kuten NinjaMock jotka mahdollistavat nopean toiminnallisen mallin luomisen tulevasta toteutuksesta. Toisaalta mallinnukseen voidaan käyttää myös paperille piirrettyä mallia. Malli on tarkoitus jättää ulkoasultaan pelkistetyksi, jotta suunnittelun fokus pysyy toiminnallisuudessa, eikä harhaudu visuaaliseen ulkoasuun tässä suunnittelun vaiheessa. Näitä malleja kutsutaan myös rautalankamalleiksi niiden yksinkertaisuuden ja helpon toteutuksen takia. (Aaronen 2017, 14.)

Yksi tärkeä osa suunnittelua on ohjelmiston visuaalisen ilmeen suunnittelu, joka määrittelee lopullisen tuotteen ilmeen. Hyvä visuaalinen toteutus ohjaa käyttäjää käyttä-

mään ohjelmistoa oletetulla tavalla. Visuaalisen suunnittelun edellytys on, että ohjelmiston alustavat rautalankamallit on suunniteltu, jotta niiden pohjalta toiminnallinen rakenne on jo alustavasti tiedossa. (Aaronen 2017, 15.)

Toiminnallinen suunnittelu kohdistuu erillisiin komponentteihin ja lopulta tarkentuu niin yksityiskohtaiseksi, että ohjelmakoodin toteuttaminen voidaan sen pohjalta aloittaa. Komponenttien suunnittelun on tarkoitus määritellä komponentin rakenne niin tarkasti, että toteutuksesta tulee laadukas ja helposti ylläpidettävä. (Haikala & Mikkonen 2011, 177.)

2.3 Toteutus

Toteutus tarkoittaa ohjelmakoodin ja siihen liittyvän dokumentaation, kuten käyttöohjeiden toteutusta.

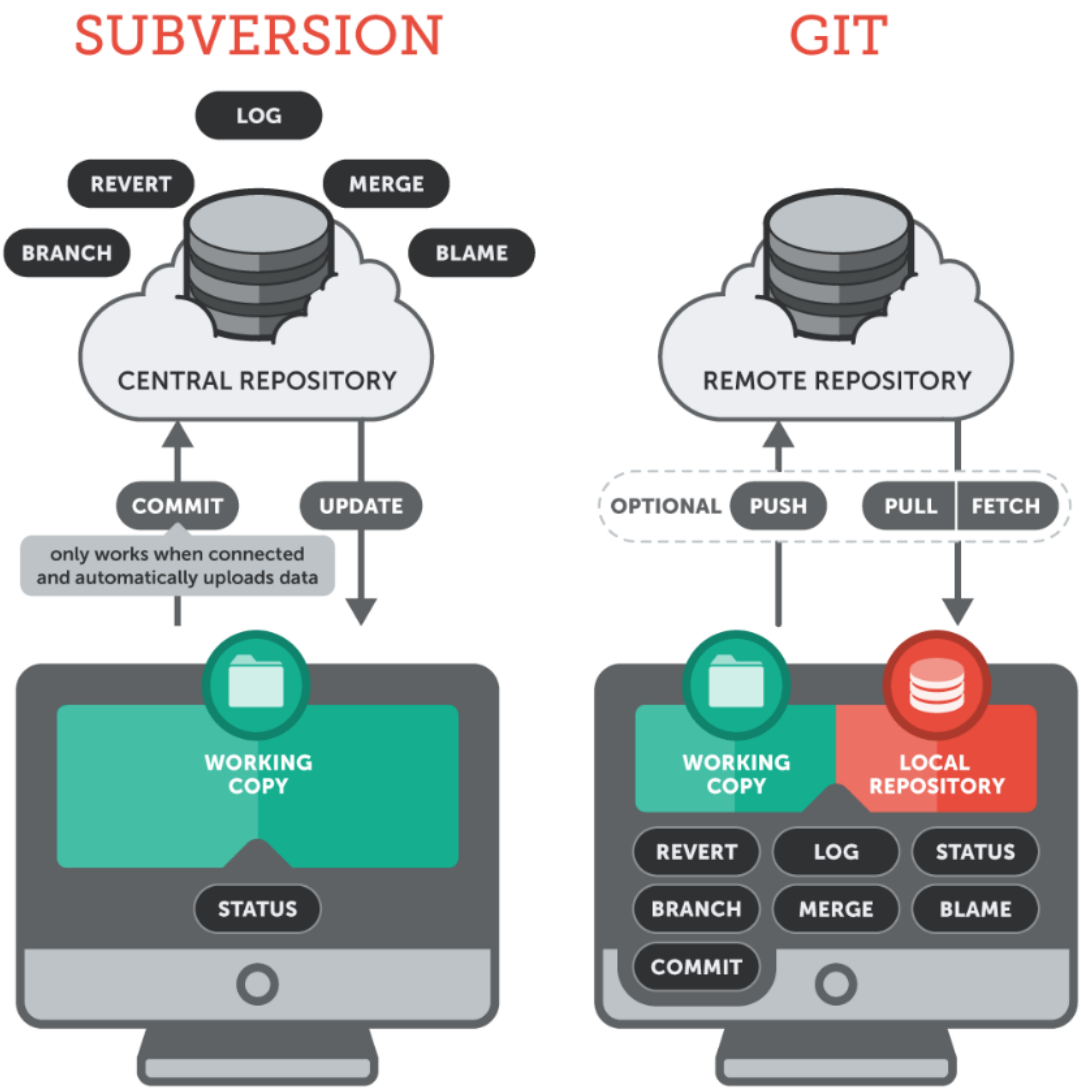
Toteutus tehdään yksityiskohtaisen suunnittelun pohjalta, jolloin laajempikin ohjelmisto on teknisesti yhtenevä ja helposti ylläpidettävä, vaikka toteutusta tehtäisiin hajautetusti useammalla tiimillä tai mahdollisesti useamman yrityksen toimesta. (Haikala & Mikkonen 2011, 183-184.)

Toteutus koostuu prototyypin iteratiivisesta toteuttamisesta, toteutuksen iteratiivisesta testauksesta, yksikkötestien luomisesta ja lähdekoodin versionhallinnasta. Lähdekoodin versionhallinta on merkittävässä roolissa ohjelmiston kehittämisessä, koska sillä mahdollistetaan usean kehittäjän yhtäaikainen lähdekoodin käsitteleminen. (Aaronen 2017, 15.)

2.3.1 Lähdekoodin versionhallinta

Ohjelmiston toteutus vaatii lähes poikkeuksetta selkeän tavan hallita lähdekoodin versiointia ja muutoksia. Versionhallinta mahdollistaa useamman kehittäjän lähdekoodin yhtäaikaisen muokkaamisen. Lähdekoodin versionhallinta voidaan jakaa kahteen eri versionhallintaan. Toinen on SVN, joka on keskitetty versionhallinta ja toinen on Git, eli hajautettu versionhallinta. Kuvio 2 kuvaa näiden kahden version eroavaisuuksia.

Merkittävin ero näissä kahdessa versionhallinnassa on hajautetun versionhallinnan paikallinen ohjelmavarasto, joka sisältää muutoshistorian ja mahdollisuuden tehdä commitit tähän paikalliseen ohjelmavarastoon, kun keskitetyssä versionhallinnassa muutoshistoria on verkon takana ja commitit tehdään verkon yli.



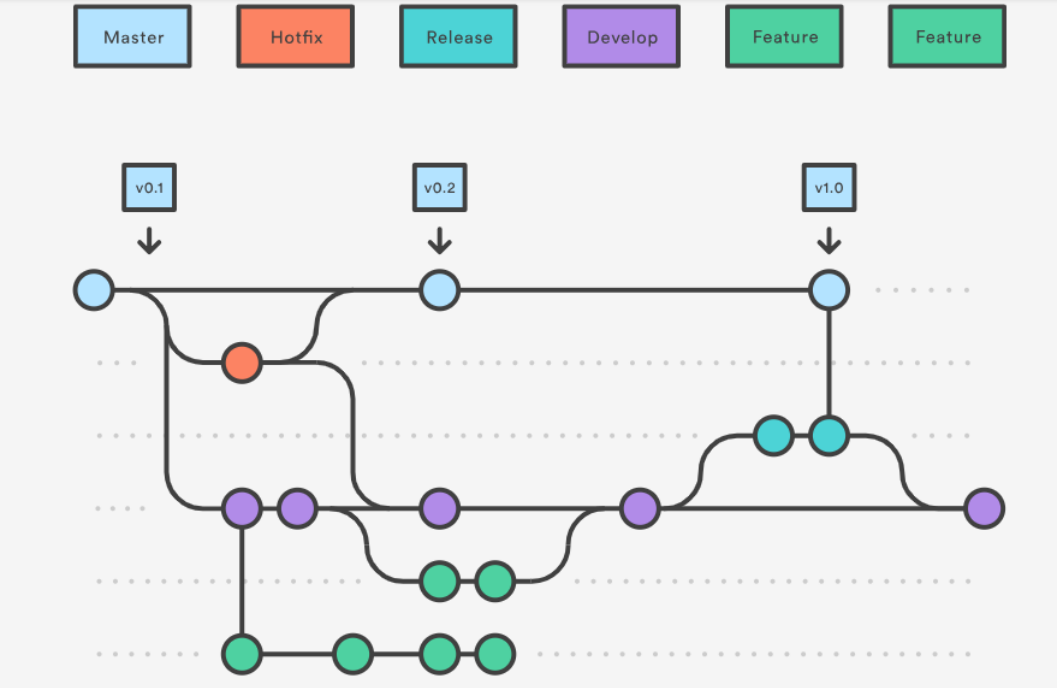
Kuvio 2. Subversion vs Git (Learn Version Control with Git N.d.)

2.3.2 Gitflow workflow

Gitflow Workflow:n ensimmäisenä esitteli Vincent Drissen. Tämä Gitin käyttö tukee sykleissä tuotettavia julkaisuja ja määrittelee tarkoin miten kutakin versionhallinnan haaraa tuotannossa käytetään. (Gitflow workflow N.d.)

Gitflow lähdekoodin versionhallinta mahdollistaa useamman toimitushaaran versionhallinnan ja helpottaa, sekä selkeyttää testaajien työtä projekteissa, joissa ylläpidetään useampaa versiota samasta ohjelmistosta. (Gitflow workflow N.d.)

Alla on selitettynä erilaiset versionhallinnan haarat ja kuviossa 3. on visuaalinen selitys haarojen relaatioista keskenään.



Kuvio 3. Lähdekoodin versionhallinta (Gitflow workflow N.d.)

Master-haara

Master-haarassa on asiakkaalle toimitettu ja toimivaksi todettu versio. Kolmannen osapuolen komponentteja voidaan testata Master haaran versioon, jonka tiedetään olevan tuotantoympäristössä ja toimivan, kun muissa haaroissa voi olla uutta julkaisematonta toteutusta samaan aikaan integroituna.

Develop-haara

Tämä haara sisältää kaiken uuden koodikatselmoidun toteutuksen. Develop-haara ei ole kehitystä varten, vaan toteutus tuodaan Develop-haaraan Feature-haaroista. Tässä haarassa toteutusta voidaan testata ja käsitellä osana kokonaisuutta. Develop-haara on aina kehityksessä edellä muita haaroja.

Feature-haara

Feature-haarassa tuotetaan lähdekoodi pieninä kokonaisuuksina ja liitetään toteutuksen jälkeen Develop-haaraan. Kehittäjä tekee Pull Requestin koodilleen, kun Feature-haaran toteutus on kehittäjän mielestä valmis. Pull Request on toiminto, joka pakottaa toisen kehittäjän hyväksymään toteutuksen ennen toteutuksen siirtämistä Develop-haaraan. Ohjelmakoodi tulee katselmoitua kattavasti, kun jokaista Feature-haarassa tehtyä toteutusta varten tehdään Pull Request, tämä taas parantaa tuotetun koodin laatua.

Release-haara

Release, eli julkaisuhaara on nimensä mukaisesti toimitettavaa julkaisua varten. Tässä haarassa tuotettu toteutus testataan ja katselmoidaan osana koko toimitettavaa järjestelmää. Tähän testaukseen kuuluu myös julkaisuun liittyvän ohjelmiston dokumentaation päivitys ja dokumentaation versionhallinnan päivitys. Asiakkaalle toimitetaan uusin versio Release-haarasta ja vasta kun toteutus on tuotantokäytössä todettu toimivaksi, se päivitetään Master-haaraan. Release haaroja voi olla myös useampia kappaleita, jolloin toteutus ei välttämättä mene suoraan Master-haaraan.

HotFix-haara

Asiakkaan tuotantokäytössä olevasta ohjelmistosta raportoidut viat korjataan HotFix-haarassa ja korjaukset liitetään sitten Release-haaraan asiakkaalle toimitettavaksi. Projekteissa, joissa toimitetaan useaa eri versiota, HotFix-haaran korjaus voidaan viedä myös Develop-haaraan, mikäli havaittu vika koskee myös muita ohjelmiston versioita.

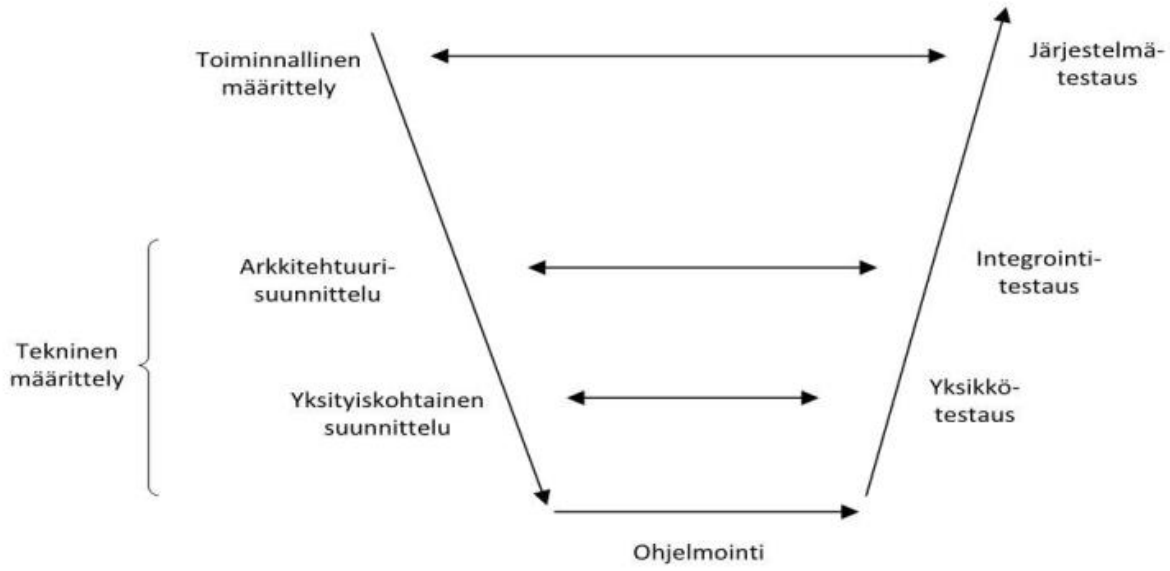
2.3.3 Yksikkötestaus osana toteutusta

Yksikkötestit ovat tärkeä osa ohjelmiston toteutusta. Kehittäjä tekee toteutuksen yhteydessä yksikkötestit aina, kun toteutus on testattavissa yksikkötestein. Kehitys voi olla myös testivetoista kehitystä, jolloin ensimmäinen suunniteltava ja toteutettava asia kehityksessä on testitapaus, jonka pohjalta itse ohjelmiston toteutus tehdään.

Yksikkötestaus eli moduulitestaus testaa pienintä mahdollista ohjelman osaa, kuten aliohjelmaa. Näiden yksikkötestien tarkoitus on varmistaa, että yksittäinen pieni osa ohjelmaa tekee juuri sen, mitä siltä odotetaan. Yksikkötestejä ajetaan usein automatisoidusti ja näillä pyritään varmistamaan, etteivät muutokset koodissa ole rikkoneet aikaisemmin tehtyä toteutusta. Yksikkötestit luo kehittäjä samalla, kun itse lähdekoodi luodaan. Yksikkötesteistä lisäksi lyhyt kuvaus kappaleessa 2.4.1.

2.4 Testaus

Testauksen tehtävä on varmistaa ohjelmakoodin tekninen toimivuus, sekä ohjelmiston vaatimustenmukaisuus. Testaukseen liittyviä vaiheita ovat suunnittelu, jossa luodaan testaussuunnitelma ja testitapaukset, testiympäristön luonti, itse testien suorittaminen ja testauksen tulosten tarkastelu. Karkeasti testit jaetaan kolmeen osioon, jotka ovat yksikkötestaus, integrointitestaus ja järjestelmätestaus. Kuviossa 4 on kuvattuna yleisesti ohjelmistokehityksessä tunnettu testauksen V-malli, joka kuvaa erilaisten projektin vaiheiden ja testauksen suhdetta keskenään. Yksi tärkeä huomio testauksesta on se, että mitä myöhäisemmässä testauksen vaiheessa viat havaitaan, sitä kalliimmaksi yleensä korjaaminen tulee. (Haikala & Mikkonen 2011, 205-208.)



Kuvio 4. V-malli (Haikala & Mikkonen 2011, 207.)

2.4.1 Yksikkötestaus/Moduulitestaus

Yksikkötestit testaavat yleensä yhtä luokkaa, joka sisältää 100-1000 riviä ohjelmakoodia. Yksikkötestit luo yleensä kehittäjä, joka luokan on toteuttanut.

Yksikkötestit vertaavat toteutusta tekniseen dokumentaatioon, kuten yksityiskohtaiseen ja arkkitehtuuriseen suunnitteludokumentaatioon. (Haikala & Mikkonen 2011, 207.)

2.4.2 Vertaisarviointi

Vertaisarviointi on niin sanottu koodikatselmointi, jossa yksi tiimin jäsen katselmoi ohjelmakoodin toteutuksen ja testaa tarvittaessa toteutuksen toiminnallisuuden kehitysympäristössä ennen kuin hyväksyy toteutuksen liitettäväksi koodin versionhallintaan.

Vertaisarvioinnin suorittaa aina joku muu, kuin henkilö, joka on kirjoittanut lähdekoodin muutoksen. Lisäksi vertaisarvioijalla pitää olla riittävä tuntemus ohjelmistosta, johon toteutus on tehty, jotta katselmointi on tarkoituksenmukainen. Koodikatselmoinnin suorittaa usein järjestelmäarkkitehti, tai vanhempi ohjelmoija.

2.4.3 Staattinen koodianalyysi

Staattisen koodianalyysin tarkoitus on havaita ohjelmakoodin virheet ja riskejä sisältävät ohjelmakoodin osat ilman koodin ”ajamista”, eli jo ennen kuin ohjelmaa käytetään. Tämä mahdollistaa virheiden havaitsemisen ja korjaamisen jo tuotantoprosessin varhaisessa vaiheessa. (Penttilä 2014, 12-13.)

Koodin laatua tarkkaillaan erilaisilla soveltuvilla ohjelmistoilla, jotka on yhdistetty Jenkinsissä, tai vastaavassa käännösympäristössä määriteltyihin tehtäviin. Nämä koodin analysointi ohjelmat ovat pääsääntöisesti projekti-, ja käännösympäristökohtaisia ohjelmia, kuten SonarQube tai Cppcheck.

2.4.4 Integraatiotestaus

Integraatiotestauksessa testataan useamman luokan yhdessä toteuttavaa osajärjestelmää. Tämä testauksen vaihe pyrkii testaamaan pääasiassa rajapintojen toimivuutta. (Haikala & Mikkonen 2011, 207-208.) Tämä testauksen osa-alue kulkee rinnan yksikkötestauksen kanssa. Kun yksikkötestejä luodaan, mietitään samalla rajapintojen testausta. Integraatiotestit laajenevat, kun valmiiden luokkien määrä kasvaa. (Haikala & Mikkonen 2011, 207-208.) Kehityksen alkuvaiheessa integraatiotestit voivat olla vielä melko yksinkertaisia, mutta ohjelmiston kasvaessa kehityksen edistyessä, myös integraatiotesteistä tulee usein laajempia ja monimutkaisempia. Integraatiotestejä ajetaan myös regressiotesteinä, jolloin aiemmin kehitettyjä testejä ajetaan aina uudestaan, kun ohjelmisto muuttuu. Integraatiotestejä ajetaan joko manuaalisesti tai automatisoidusti. (Testauksen tasot N.d.)

2.4.5 Järjestelmätestaus

Järjestelmätestaus on nimensä mukaisesti kattava koko järjestelmää koskeva testaus, joka voi sisältää esimerkiksi kenttätestauksen, hyväksymistestauksen, kuormitustestauksen, luotettavuustestauksen, asennustestauksen, käytettävyystestauksen ja regressiotestauksen. (Haikala & Mikkonen 2011, 208.)

Kenttätestaus on loppukäyttäjän näkökulmasta aidossa käyttöympäristössä tehtävää testausta, joka todentaa järjestelmän vaatimusten mukaisuuden (Tersa 2002, 62.) Hyväksymistestaus on yleensä asiakkaan toteuttamaa testausta, mutta toimittaja on tässä usein mukana tukemassa testausta (Tersa 2002, 23.) Kuormitustestaus taas kertoo, miten hyvin järjestelmä sietää normaalin käytön aiheuttamaa kuormaa ja miten järjestelmä reagoi normaalia suurempaan kuormaan. Luotettavuustesteillä selvitetään miten hyvin järjestelmä toipuu virhetilanteista ja miten virheherkkä järjestelmä on. Asennustesteillä varmistetaan että asennus toimii odotusten mukaisesti. Käytettävyystestit taas osoittavat miten hyvin järjestelmän ominaisuuksia voidaan käytännössä käyttää. Regressiotestejä voidaan ajaa kaikilla testauksen tasoilla. Nämä testit varmistavat, että järjestelmään tehdyt muutokset eivät ole rikkoneet muuta ohjelmiston toiminnallisuutta. (Haikala & Mikkonen 2011, 208.)

2.4.6 Laadunvarmistus

Laadunvarmistuksen tarkoitus on havaita virheet tuotannossa ja kattaa kaikki menetelmät, joilla virheitä voidaan vähentää. Yksi iso tekijä virheiden ehkäisyssä on standardienmukaiset prosessit. Osa testauksesta kuuluu laadunvarmistukseen, kuten ohjelmiston teknisten ominaisuuksien, esimerkiksi suorituskyvyn testaus on osa laadunvarmistusta. (Haikala & Mikkonen 2011, 137-150.)

2.4.7 Ei-toiminnalliset testit

Ei-toiminnalliset testit ovat esimerkiksi käytettävyyteen, standardeihin ja ylläpidettävyyteen liittyviä testejä, jotka suunnitellaan järjestelmän ei-toiminnallisten vaatimusten pohjalta. Nämä testit kohdistuvat usein koko järjestelmään, eikä pelkästään yhteen toteutettuun toiminnallisuuteen.

2.5 Modernit tuotantomenetelmät

DevOps tulee englanninkielisistä sanoista Development ja Operations, eli kehitys ja tuotanto. DevOps on ketterien menetelmien tarpeisiin luotu ympäristö jonka tarkoitus on mahdollistaa ohjelmistojen julkaiseminen tuotantoon nopeasti. Kaksi DevOpsin kannalta tärkeintä käytäntöä ovat jatkuva integraatio ja jatkuva toimitus.(DevOps N.d.)

2.5.1 Jatkuva integraatio

Jatkuvassa integraatiossa ohjelmisto integroidaan ja koostetaan jatkuvasti. Tämä on toteutettu siten, että jokaisen muutoksen jälkeen ohjelmisto koostetaan. Jos ohjelmisto on laaja ja toteutusta tehdään usean kehittäjän, tai jopa usean toimittajan toimesta, voi olla kannattavaa koostaa ohjelmisto ajastetusti esimerkiksi öisin, jolloin koostaminen ei hidasta toteutusta. Perinteisemmissä tuotannon prosesseissa tämä ohjelmiston integraatiotyö sijoittuu projektin loppuun, jolloin kehityksen aiheuttamat muutokset eri komponenttien välillä tulevat ilmi vasta projektin lopussa. (DevOps N.d.)

2.5.2 Jatkuva toimitus

Jatkuva toimitus vaatii yritykseltä selkeää menettelytapaa, jolla lähdekoodi käsitellään, ennen kuin ohjelmisto voidaan julkaista asiakkaalle. Jatkuva toimitus on jatkoa jatkuvalle integraatiolle ja on tärkeä osa ketterää ohjelmistokehitystä.

Jatkuva toimitus tarkoittaa sitä, että on aina saatavilla uusin automatisoidut testit läpäissyt versio ohjelmistosta käännettynä. Tarvittaessa käännökset voidaan ajaa myös manuaalisesti. (DevOps N.d.)

3 Ohjelmistotuotannon prosessimallit

Ohjelmistotuotannossa käytettiin pitkään vesiputousmallia, joka on toimiva malli vielä nykyäänkin kehitystöissä. Nykyisin suuren suosion on saanut ketterät kehitysmallit.

3.1 Vesiputousmalli

Vesiputousmallin mukaan mennään vaihe kerrallaan loppuun ja kun yksi vaihe on saatu hyväksytysti loppuun, siihen ei enää palata uudestaan. Aiemmin kappaleessa 2 esitelty Kuvio 1 kuvaa perinteistä vesiputousmallia ja sen vaiheistusta.

3.2 Ketterät kehitysmallit

Nykyisin ohjelmistokehityksessä on pääsääntöisesti siirrytty käyttämään yleisesti tunnettua ketterää kehitystä, jotka perustuvat PDCA-malliin, joka on Kuvio 5 kuvattuna.

Ketterä kehitys sai nykyiset ketterän ohjelmistokehityksen viitteet vuonna 2001, kun Agile Manifesto luotiin 17 kevyen kehityksen asiantuntijan toimesta. (Cunningham 2001.)

Nykyisin yhä laajemmin on otettu käyttöön mukautettuja ketterän kehityksen metodeja, joissa otetaan tiimin käyttöön sopivat toimintametodit ja jätetään pois se, mikä aiheuttaa liikaa haasteita verrattuna hyötyyn.

Kuvio 5. PDCA-ympyrä (Viisi kysymystä N.d.)

3.2.1 The Agile Manifesto

The Agile Manifesto luotiin 2001 Utahissa. 17 kevyiden kehitysmenetelmien asiantuntijaa kokoontui laskettelemaan, rentoutumaan ja keskustelemaan ohjelmistokehityksen yhteisistä käytänteistä. Tämä 17 henkilön ryhmä nimesi itsensä The Agile Alliance nimiseksi ryhmäksi, eli ketteräksi liittoumaksi. Agile Manifeston periaatteet pätevät kaikessa ketterässä kehityksessä. Ketterä kehitys perustuu neljään arvoon ja 12 periaatteeseen, jotka 2001 määriteltiin (Cunningham 2001).

Ketterien menetelmien arvot Agile Manifeston mukaan (Cunningham 2001) ovat:

- *Yksilöiden ja vuorovaikutuksen arvostus ennen prosesseja, sekä työkaluja*
- *Toimivan ohjelmiston, tai sovelluksen arvostus ennen kokonaisvaltaista dokumentaatiota*
- *Asiakasyhteistyön arvostus ennen sopimusneuvotteluja*
- *Muutokseen reagoiminen ennen suunnitelman noudattamista*

Agile Manifestissa(Cunningham 2001). määritellyt periaatteet ovat:

- *Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.*
- *Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.*

- *Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.*
- *Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.*
- *Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.*
- *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*
- *Toimiva ohjelmisto on edistymisen ensisijainen mittari.*
- *Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.*
- *Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.*
- *Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.*
- *Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.*

3.2.2 Lean

Lean on Japanissa alun perin teollisuuteen kehitetty tuotantotapa, joka pyrkii tehostamaan tuotantoa poistamalla tuotannosta turhaa ja hukkaa ajankäytössä sekä materiaalissa. Lean periaatteita on alettu käyttää myös ohjelmistokehityksessä tavoitteena tehdä prosesseista tehokkaampia. Isoin haaste Leanin periaatteiden toteuttamisessa ohjelmistokehityksessä on mitata oikeita asioita prosessissa. Teollisuudessa hukan mittaaminen ja sen kautta prosessin kehittäminen on helpompaa, koska voidaan tarkkailla fyysistä materiaalia, sekä tuotteen läpimenoaikaa. Ohjelmistokehityksessä tuotanto on luovaa ja pääsääntöisesti ihmisen toteuttamaa. Toisin kuin teollisuudessa, jossa kappaleita usein tuotetaan suuria massoja ja toteutus tapahtuu automatisoiduilla laitteilla, joita ihminen vain ohjaa, ohjelmistokehityksessä tuotteita on usein vain yksi. (Tyynismaa 2014, 37-40.)

Lean-ohjelmistokehityksestä ei ole yhtä oikeaa toteutustapaa, vaan se perustuu lähinnä Lean-ajatteluun ja jatkuvaan kehittämiseen. Kaksi tulkintaa Lean-ohjelmistokehityksestä Mary ja Tom Poppendieckien tulkinta ja Womackin ja Jonesin tulkinta, ovat paljon viitattuja tulkintoja. (Tyynismaa 2014, 37-40.)

Poppendieckit loivat Lean-ohjelmistokehitykseen seitsemän soveltuvaa periaatetta:

- Poista hukka
- Laadun rakentaminen ohjelman sisään
- Tietämyksen kehittäminen
- Sitoutumisen lykkääminen
- Nopea toimitus
- Ihmisten kunnioitus
- Kokonaisuuden optimointi

Womackin ja Jonesin tulkinta pohjautuu viiteen toimialasta riippumattomaan periaatteeseen, jotka soveltuvat myös Lean-ohjelmistokehitykseen:

- Arvo
- Arvovirta
- Virtaus
- Imu
- Täydellisyys

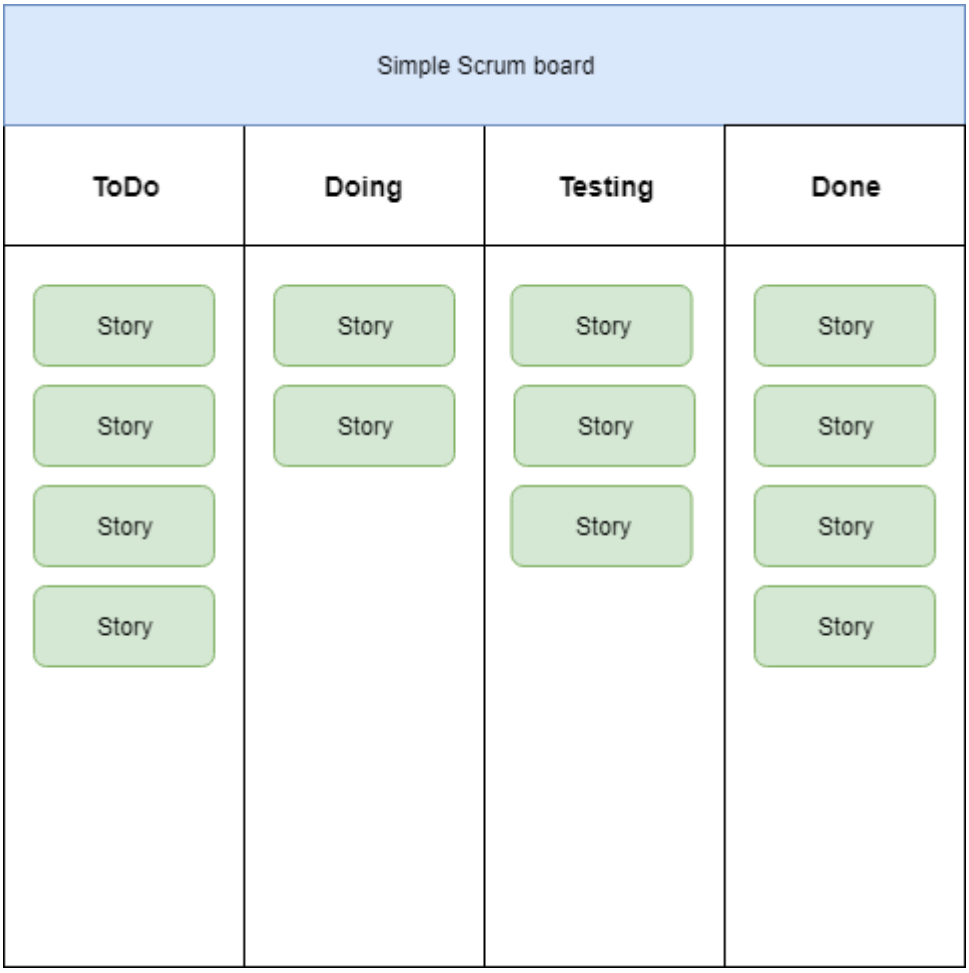
(Tyynismaa 2014, 37-40.)

3.2.3 Scrum

The Scrum guiden mukaan Scrumin tehokkuus perustuu todellisten tulosten analysointiin kokemuksen perusteella eikä ennalta laaditun fiktiivisen suunnitelman orjalliseen noudattamiseen (The Scrum guide 2017, 4.)

Scrum määrittelee tiimin roolit, säännöt ja sprinttien, eli kehityssyklien toiminnat sääntöineen tarkasti. (The Scrum guide 2017, 3.)

Scrum-tiimi jakaa kehityksensä kahdesta neljään viikkoon kestäviin sykleihin, joita kutsutaan Sprinteiksi. Näihin Sprintteihin tiimi valitsee aina työtä, joka on sprintin jälkeen valmiina julkaistavaksi. Kuviossa 6 on kuva yksinkertaisesta Scrum-taulusta kehityksen aikana.



Kuvio 6. Yksinkertainen Scrum-taulu

Sprinttien suunnittelu (Sprint Planning) on jaettu kahteen erilliseen osioon, joista ensimmäisessä tuotteen omistaja (Product Owner) priorisoi backlogia, jolla kaikki toteutusta odottavat tehtävät ovat. Toisessa vaiheessa tiimi ja Scrum Master, joka valvoo Scrumin periaatteiden noudattamista tiimissä, valitsevat yhdessä sopivan määrän toteutusta seuraavan sprintin sisällöksi niin, että valittu toteutus tulee valmiiksi sprintin päättyessä. (The Scrum guide 2017, 10-11.)

Päivittäisellä tasolla tiimi kokoontuu Daily Scrum -kokoukseen samaan paikkaan kerran päivässä keskustelemaan edellisen ja tulevan päivän tekemisistään. Näissä päivittäisissä tapaamisissa voidaan myös keskustella vastaan tulleista haasteista tiimin kanssa. (The Scrum guide 2017, 12.)

Sprintin päätteeksi tiimi pitää kaksi palaveria, joista ensimmäinen on Sprint Review, jossa tiimi käy sprintin aikana valmistunutta toteutusta läpi yhdessä. Toinen palaveri

on Sprint Retrospective, jossa taas tarkastellaan itse prosessia, eli mikä edellisessä Sprintissä onnistui hyvin ja missä olisi parantamisen varaa. (The Scrum guide 2017, 13-14.)

Scrumin menetelmä perustuu viiteen eri arvoon, jotka ovat sitoutuminen, fokusointi, avoimuus, kunnioitus ja rohkeus. (The Scrum guide 2017, 5.)

Sitoutuminen:

Scrum-tiimin täytyy sitoutua asettamaan realistiset tavoitteet ja onnistua saavuttamaan tavoitetila tiiminä (The Scrum guide, 2017, 5). Scrum menetelmä pilkkoo toteutuksen tietyn mittaisiin aikasykleihin, joiden sisällä tehtävän toteutuksen tiimi pilkkoo riittävän pieniksi kokonaisuuksiksi, jotta työ on tehokasta ja etenemistä voidaan seurata päivittäin. Tällä tavoin Scrumin avulla tiimi saavuttaa asettamansa tavoitteet.

Fokusointi:

Fokusointi Scrumissa tarkoittaa sitä, että keskitytään ainoastaan muutamaan asiaan kerrallaan. Kehitysjaksoittain päätetään tiimin tavoitteet, joihin sitoudutaan ja keskitytään jakson aikana. (Layton 2015.)

Avoimuus:

Kaikki projektissa on avointa ja vapaasti tarkasteltavissa sekä paranneltavissa. Scrum menetelmä vaatii, että jokaisen projektiryhmän jäsenen tavoitteet ovat kaikkien projektiryhmän jäsenien tiedossa ja nähtävillä. (Layton 2015.)

Kunnioitus:

Jokainen tiimin jäsen valitaan vahvuuksiensa vuoksi, tämän mukana tulevat myös heikkoudet ja mahdollisuus oppia ja kehittyä tiimin jäsenenä. Scrumin kultainen sääntö on, että jokaisen tiimin jäsenen tulee kunnioittaa toistaan. Scrumissa ei pidetä yksilöä vastuussa toteutuksesta, vaan ainoa toteutuksesta ja tavoitteista vastuussa oleva on tiimi kokonaisuutena. Näin toimittaessa yhteistyö tulee olemaan tiimin sisällä parhaimmillaan. (Layton 2015.)

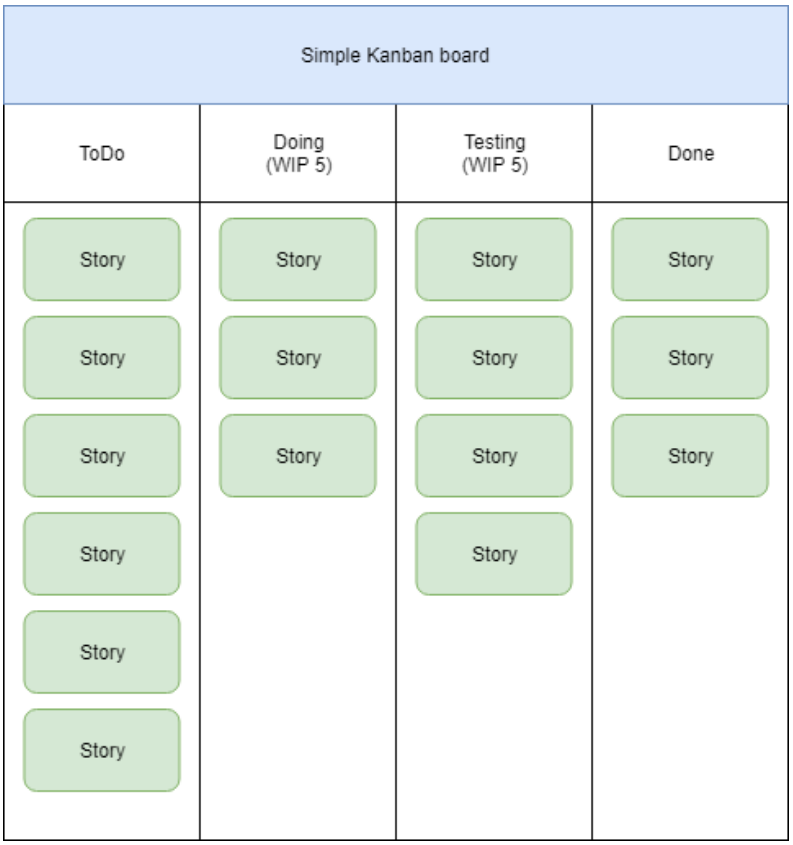
Rohkeus:

Scrumin avoimuus vaatii tiimin jäseniltä rohkeutta myöntää omat virheensä, antaa avointa palautetta ja toimia tiimin päätösten mukaan. (Layton 2015.)

3.2.4 Kanban

Kanban on Toyotalla kehitetty tuotannon menetelmä, joka keskittyy työmäärän ja työn priorisointiin sekä tehokkuuden optimoimiseen. Ohjelmistoalalla tämä tarkoittaa sitä, että kehitystiimi ottaa toteutuksen alle sopivan määrän työtä kerrallaan ja tekee toteutuksen loppuun, ennen kuin ottaa lisää työtä tehtäväkseen. (Majchrzak, M. 2017, 40-41.)

Kanban on hyvin avoin menetelmä ja toisin kuin Scrum se ei ota kantaa tiimin roolitukseen, eikä ole sidottuna aikaan perustuviin sykleihin. Kanban ottaa kantaa siihen, minkä verran työtä on kerrallaan työn alla, jolloin tuotantoprosessin ongelmakohdat tulevat nopeasti esiin. Ongelmakohtina voidaan pitää sellaisia prosessin kohtia, joissa tehtävät työt viettävät muita prosessin kohtia pidempään aikaa, ennen kuin siirtyvät seuraavaan prosessin kohtaan. Kanban-taulu yksinkertaisimmillaan on kuvattu kuviossa 7.



Kuvio 7. Yksinkertainen Kanban-taulu

Itse tuotantoprosesseihin Kanban ei suoranaisesti ota kantaa eikä määrittele ennalta tapoja muuttaa tekemistä, vaan odottaa jokaisen projektiin osallistuvan yksilön tuovan omat kehitysehdotuksensa esille. Ohjelmistoalalla kehittäjillä on selkein kuva siitä, miten ohjelmistotuotannon prosessia pitäisi parantaa. Kanban menetelmä edellyttää hallinnollisen tason tukevan näitä kehitysehdotuksia ja näin prosesseja voidaan kehittää pikkuhiljaa ja muutos on jatkuvaa. (Leopold 2015, 11-16.)

Kanbanin kuusi pääperiaatetta Leopoldin (2015, 18) mukaan Anderson kertoo olevan visualisointi, työmäärän rajoittaminen, työnkulun hallinnointi, tuotantoprosessin määrittely, palautteen anto ja yhteistyössä parantaminen. (Leopold 2015, 40-41.)

Visualisointi:

Visualisointi tulisi aloittaa kokonaiskuvalla, jonka avulla nähdään koko tuotantoprosessi alusta loppuun. Tämän kuvauksen avulla voidaan miettiä mistä kohtaa ketjua Kanbanin metodeja ryhdytään toteuttamaan ja muokkaamaan, sekä tunnistaa eri ketjun osien rajat kokonaiskuvassa. (Vacanti 2018, 5.)

Seuraavaksi tulisi visualisoida erilliset prosessit ja tunnistaa todellinen käytössä oleva prosessi sen sijaan, että kirjoitetaan teoreettinen prosessin mukainen kuvaus. Yleensä käytännön toteutus poikkeaa kirjoitetusta prosessista joiltain osin. Prosessin muutos parempaan Kanbanin avulla on hitaampaa, jos tässä kohtaa ei visualisoida käytössä olevaa realistista prosessia. (Leopold 2015, 18-19.)

Työnkulun lisäksi pitää vielä visualisoida itse työ. Yleisin tapa on käyttää niin sanottuja Post-it lappuja, joita on myös visualisoitu jo lähes kaikissa tuotannonseurantaan käytettävissä työkaluissa. Tähänkään visualisointiin ei ole yhtä oikeaa tapaa, vaan tiimin tulee löytää oma tapansa visualisoida työ riittävän hyvin, jotta itse kehittäminen kuvauksen perusteella olisi helppoa. (Leopold 2015, 18-19.)

Työmäärän rajoittaminen ja hallinnointi:

Työn alle otettavan työmäärän rajoittamisella vältytään hautaamasta hyväksyttyjä asiakkaan toivomia kehitysehdotuksia toteutusta odottavalle listalle kuukausiksi, tai jopa vuosiksi. (Leopold 2015, 19-21.)

Asiakkaan tulee priorisoida tehtävää työtä aika ajoin, kun toteutettavaksi ei oteta liikaa työtä kerrallaan ja rajoitukset koskevat myös toteutukseen hyväksyttyjä kehitysehdotuksia. Näin asiakas saa kehityksestä aina parhaimman mahdollisen hyödyn.

Työnkulun hallinta:

Kanbanin yksi tärkeimmistä tarkasteltavista asioista on työnkulun hallinta, joka keskittyy löytämään prosessin ongelmakohtia ja pullonkauloja, jossa yksittäiset työt ruuhkautuvat yhteen prosessin vaiheeseen, näin hidastaen kokonaisprosessia. (Leopold 2015, 20.)

Työnkulun hallinnalla voidaan luoda tavoitteita ja tarkastella uusien toimintatapojen vaikutusta prosessiin. Tehtyjen muutosten vaikutuksia pitää pystyä mittaamaan ja raportoimaan. Tarkoituksena ei ole tarkastella yksilötason suoritusta, vaan prosessin menetelmien tehokkuutta ja kehittää prosessia, eikä vertailla yksilöitä toisiinsa. (Leopold 2015, 20.)

Tuotantoprosessin määrittely:

Kun tiimillä on selkeät määritellyt prosessit, joiden mukaan työ etenee todellisuudessa, eikä vain suunniteltu prosessikuvaus, jota ei noudateta, voi tiimi muokata prosessia tehokkaammaksi. Mikäli prosessien kuvaukset eivät vastaa todellisuutta, on tiimien haasteellista nähdä prosessin todelliset ongelmakohdat ja prosessien kehittäminen viivästyy. (Leopold 2015, 21-22.)

Palautteen anto:

Palautteen anto on Kanban toimintamallissa erittäin tärkeä osa prosessia, koska oikeaoppinen Kanban perustuu jatkuvaan kehittämiseen ja kehittymiseen. Palautteen avulla voidaan tunnistaa prosessien kehityskohteet ja kehitettävät toimintatavat. Tehokkaimmillaan palautetta jaetaan myös hallinnon tasolla, eikä ainoastaan tiimin kesken, jolloin kehitystä voidaan toteuttaa koko tuotantoprosessissa. (Leopold 2015, 22.)

Yhteistyössä kehittäminen:

Tämän ajatuksen pohjana on tunnistaa toimintatapojen kehittämisessä jo aiemmin käytössä olleita toimintatapoja, joissa ongelmat on tunnistettu. Aina ei ole välttämättä tehokkainta kehittää uusia toimintatapoja, tai toimintamalleja vaan arvioida erilaisten olemassa olevien toimintatapojen hyötyjä, sekä haittoja. (Leopold 2015, 22-23.)

3.2.5 Scrumban

Tämä on yhdistelmä Scrumista ja Kanbanista, jossa halutessaan voidaan käyttää Scrumin aikaan sidottuja kehitys syklejä, eli Sprinttejä, mutta työn seuranta tapahtuu Kanban-taululla. Tämän menetelmän tarkoitus on yhdistää parhaat puolet näistä kahdesta menetelmästä ja sen alkuperäinen idea oli toimia siirtymävaiheena Scrumista Kanban menetelmään. (Huttunen 2017, 15-17.)

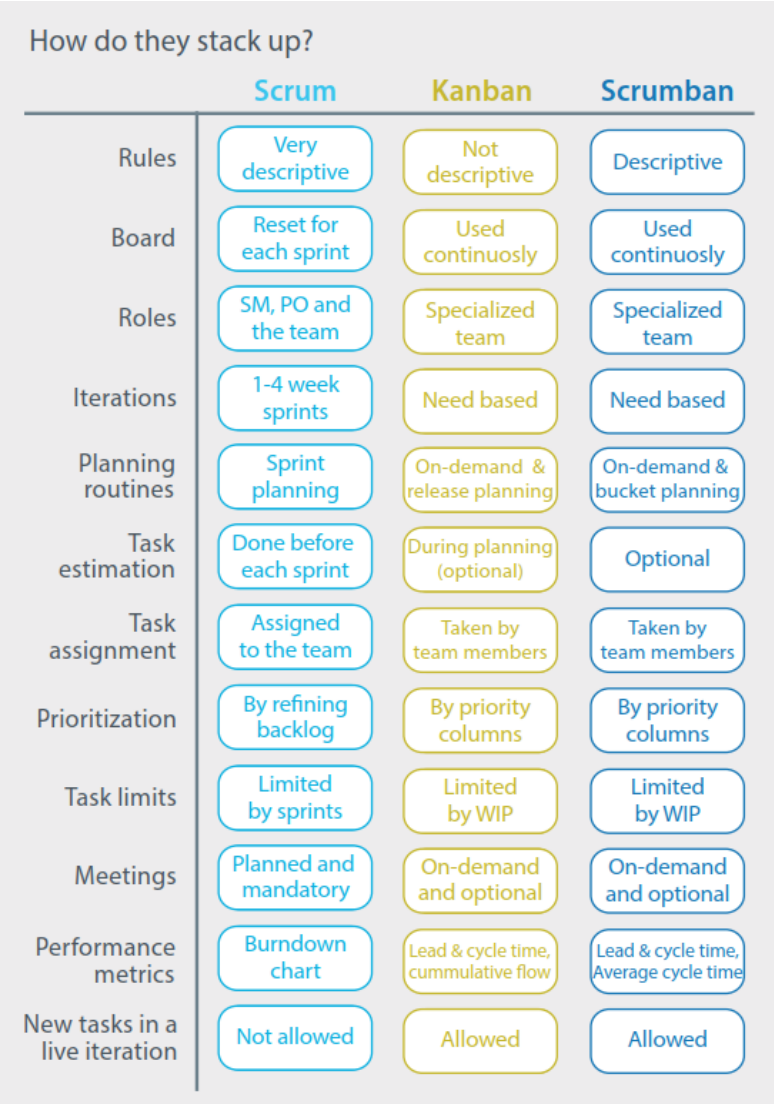
Scrumbanissa ei ole virallista toteutusta odottavien tehtävien listaa (backlog), vaan työtä pyritään määrittelemään ja suunnittelemaan juuri sopiva määrä toteutettavaksi seuraavaan julkaisuun, jonka jälkeen priorisoidaan, määritellään ja suunnitellaan lisää toteutusta työn alle. Scrumbanin rooleja on ennalta määritelty vain tiimi, mutta tuotannon vaatimat muut roolit voidaan määritellä haluamallaan tavalla. Scrumin lailla Scrumbanissa on päivittäiset tiimin lyhyet tapaamiset käytössä ja muita Scrumista tuttuja kokouksia, tai palavereita pidetään tarpeen mukaan, eikä ennalta määrättyinä prosessin ajankohtina kuten Scrumissa tehdään. Scrumbania voidaan käyttää Scrum-tiimin, eli moniosaavan tiimin kanssa, tai Kanban-tiimin kanssa, joka koostuu usein erikoisosaamista omaavista yksilöistä. Scrumban soveltuu niin ylläpidossa oleville projekteille, kuin suurille projekteille. (Pahuja N.d.)

Laajojen kokonaisuuksien suunnittelu tehdään Scrumban suunnitteluämpäreiden avulla jaksoittain vuoden, puolen vuoden ja kolmen kuukauden ämpäreissä, jolloin jokainen toteutus käy nämä kolme eri ämpäriä läpi, ennen kuin pääsee tiimin kehitettäväksi. Vuoden ämpärissä määritellään pitkäaikaiset tavoitteet. Kun tavoitteet on asetettu, voidaan siirtyä puolen vuoden ämpäriin, jossa määritellään vaatimukset toteutukselle. Viimeisessä kolmen kuukauden ämpärissä luodaan kehitystiimin työtehtävät aiemmin tunnistettujen vaatimusten pohjalta. (The-Ultimate-Agile-Guide N.d.) Kehitystiimin kannalta on tärkeää, että toteutusta koskevat vaatimukset on selvitetty

riittävällä tarkkuudella ennen kuin itse ohjelmakoodia ryhdytään tuottamaan. (Huttunen 2017, 15-17.)

Itse toteutuksessa Scrumbanissa käytetään Kanban-taulua vastaavaa työn kulun visualisointia, joka yksinkertaisuudessaan antaa nopealla tarkastelulla selkeän kuvan töiden tilanteesta kyseisellä ajanhetkellä. (Huttunen 2017, 15-17.)

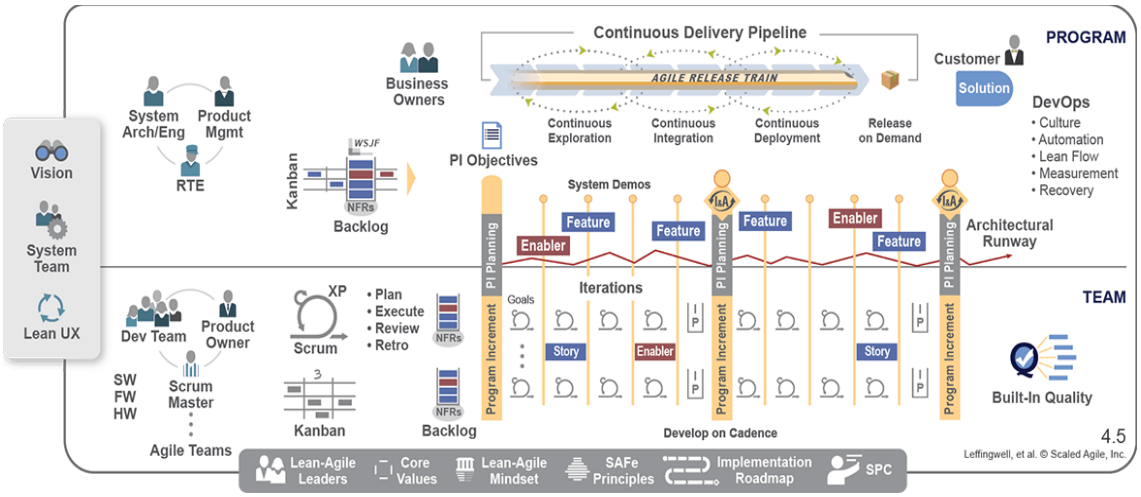
Kuviossa 8 on kuvattu kolmen yleisesti käytetyn ketterän kehityksen toimintatavan eroavaisuuksia.



Kuvio 8. The-Ultimate-Agile-Guide (The Ultimate Agile Guide N.d, 12.)

3.2.6 SAFe

SAFe, eli Scaled Agile Framework on korkean tason ketterän kehityksen viitekehys, joka antaa valmiita työskentelymalleja, sekä prosessikuvauksen tehokkaalle skaalautuvalle ketterälle kehitykselle. Kuviossa 9 on kuvattu kevyt Essential SAFe, jonka pohjalta SAFea on hyvä lähteä käyttämään. Kuvion 9 alla on kuvattu kymmenen Essential SAFe:n elementtiä. (Essential SAFe 2017.)



Kuvio 9. Essential SAFe (Essential-SAFe 2017).

1. Lean-ketterät periaatteet:

SAFe:n tarkoitus on olla kehittyvä ketterä toimintamalli, jonka periaatteet ajavat kehitystä mahdollisimman lyhyeen toimitusaikaan ja näin lisää laatua ja arvoa toteutukselle. (Essential SAFe 2017.)

2. Oikeat ketterät tiimit sekä ART:

Ketterät tiimit sisältävät kaiken tarvittavan osaamisen, mitä ohjelmistokehityksessä tarvitaan. ART, eli Agile Release Train on ketterä julkaisuprosessi, joka on pitkälle automatisoitu. Nämä yhdistettynä hyvään asiakaskommunikaatioon ovat hyvä tapa tehostaa tuotantoprosessia. (Essential SAFe 2017.)

3. Rytmi ja synkronointi:

Rytmi antaa jatkuvuutta ja tasaisuutta kehitysprosessille ja luo tarvittavat rutiinit kehitykselle. Synkronointi taas antaa prosessille useamman näkökulman ratkaisuille.

Synkronoinnilla on suuri merkitys järjestelmän osien yhteen kokonaisjärjestelmään liittämisessä. (Essential SAFe 2017.)

4. Ohjelmistokehityssyklin suunnittelu:

Kasvokkain käytävät tuotannon palaverit ovat tehokkain tapa ratkaista eteen tulevia ongelmia ja ne ovat yksi ketterän julkaisuprosessin tärkeimmistä osista. (Essential SAFe 2017.)

5. DevOps ja julkaisukyky:

DevOps tuo mukanaan automaation, turhan työn poistamisen (Lean), mittaroinnin ja kehityksen palautuspisteet, jolloin ohjelmistokehitys on tehokasta ja nopeaa. Julkaisukyky keskittyy luomaan asiakkaalle lisää arvoa nopeammin ja ketterämmin. (Essential SAFe 2017.)

6. Systeemitason Demo:

Tämän demon tarkoitus on esitellä kahden viikon välein kaikkien tiimien systeemille tekemä toteutus integroituna, jolloin projektin omistajat voivat katselmoida toteutusta osana kokonaisuutta ja tehdä muutoksia kehityksen suuntaan tarvittaessa. (Essential SAFe 2017.)

7. Tarkastele ja sopeudu:

Tämä on tilaisuus, jossa tarkastellaan ohjelmiston kehitystä 8-12 viikon jaksojen päätteeksi, näissä tilaisuuksissa esitellään ja arvioidaan sen hetkinen kehityksen tilanne. Tämän jälkeen tiimit tunnistavat parannuskohteet ja haasteet, jotka nostetaan työlistalle seuraavaa kehityssykliä varten. Myös hallinnollisia tehtäviä tekevien henkilöiden tulee pitää näitä palavereja, jotta projekteista saadaan paras mahdollinen hyöty asiakkaalle. (Essential SAFe 2017.)

8. Innovointi- ja suunnitteluiteraatio:

Kehitys voi olla tiimille hyvin intensiivistä ja aikataulut ja muut ulkopuolelta tulevat paineet voivat estää tiimiä olemasta innovatiivinen, joten SAFe:n on erikseen määritelty sykli innovointia ja uusia kehityskohteita varten. Tämä edistää toteutuksen arvoa asiakkaalle. (Essential SAFe 2017.)

9. Arkkitehtuurinen juoksurata

Tämä juoksurata tarkoittaa arkkitehtuurin suunnittelun ”polkua”. Arkkitehtuuriset muutokset pyritään laajemmissa ohjelmistokokonaisuuksissa suunnittelemaan juuri riittävästi, jotta ohjelman arkkitehtuuri ei rajoita kehitystä, mutta samalla varmistaen, ettei liian pitkälle etukäteen suunniteltu arkkitehtuuri rajoita kehitystä. (Essential SAFe 2017.)

10. Lean-Ketterä Johtajuus

Hallinnon työntekijöiden on kouluttauduttava Lean-ajattelumalliin, ketterään kehitykseen ja toteutettava projektin hallintaa ketterien ja tehokkaiden menetelmien avulla, sekä koulutettava tiimiä ketterissä menetelmissä. Vain tällä tavoin SAFe:n käyttö on tehokasta ja siitä saadaan täysi hyöty irti. (Essential SAFe 2017.)

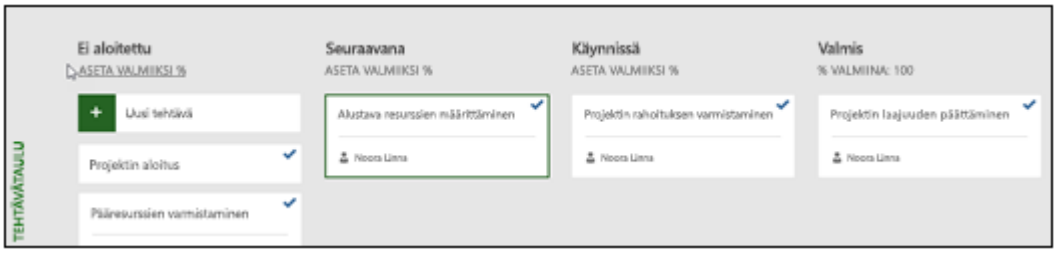
4 Ohjelmistotuotannon työkalut

Erilaisia työkaluja ohjelmistotuotannon prosesseihin tarjoavat muun muassa Microsoft, IBM, VersionOne ja Atlassian. Tässä luvussa esitellään Microsoft Projectin tarjoamia vaihtoehtoja, IBM Rationalin tarjoamia työkaluja, sekä VersionOne:n vastaavia työkaluja. Luvussa 5. käsitellään erikseen tarkemmin Atlassianin tuotteita, jotka ovat käytössä Combitech Oy:n tuotantoprosessissa.

4.1 Microsoft Project

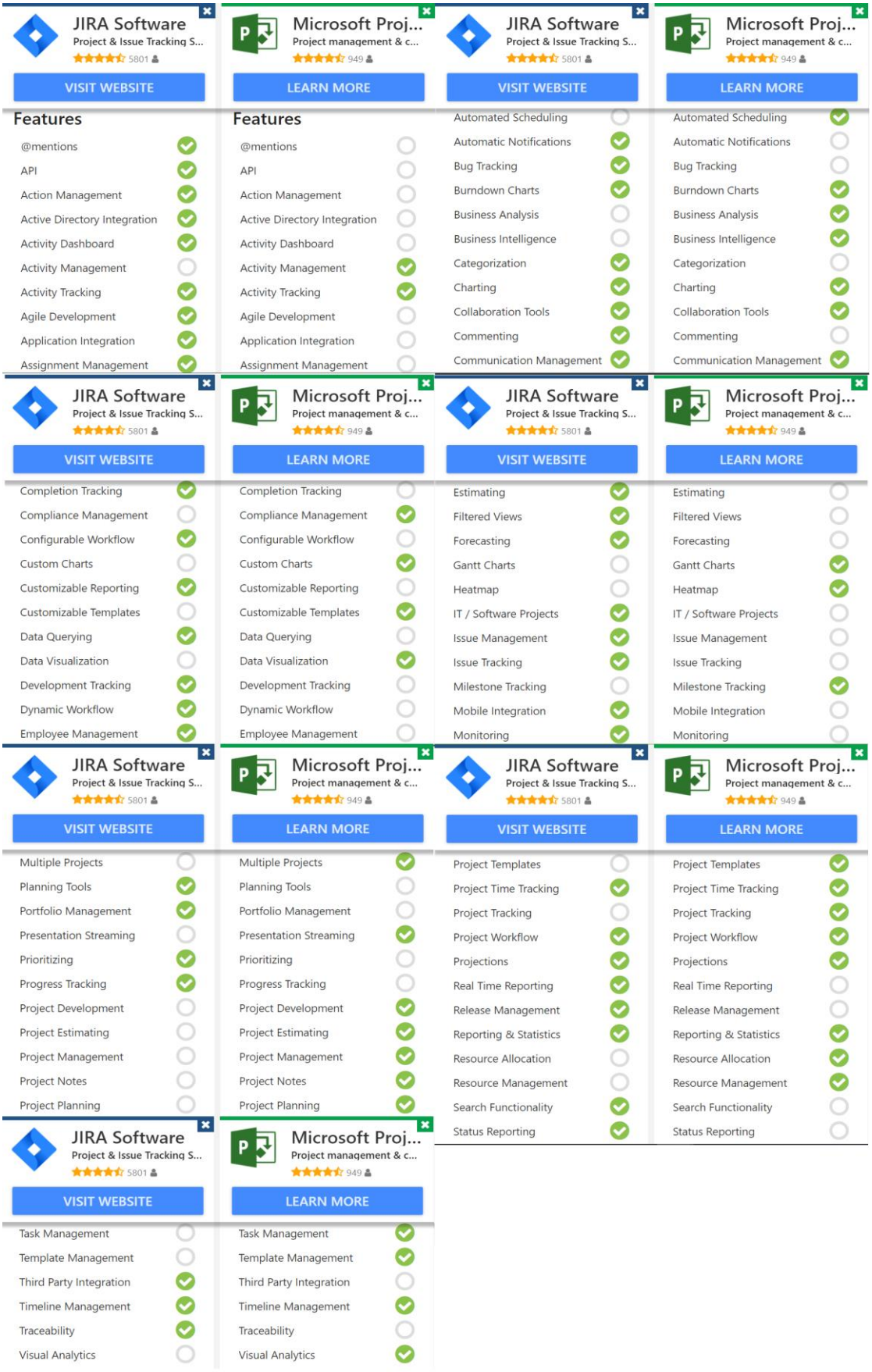
Microsoft Project on projektinhallintaan tarkoitettu työkalu, jonka saa joko pilvipohjaisena ratkaisuna, tai paikallisesti asennettavana ratkaisuna. Microsoft Projectissa on valmiita pohjia aikataulutukeen ja raportointiin. Ohjelmistolla voidaan toteuttaa portfolion hallintaa ja se on integroitavissa Microsoftin Power BI:hin, joka ovat tarkoitettu liiketoiminnan analysointiin ja visualisointiin. (Microsoft Project N.d.)

Microsoft Project Online työpöytäsovelluksessa on saatavilla myös nykyisin laajalti käytössä olevat tehtävätaulut. Kuviossa 10 on kuvattu tehtävätaulu



Kuvio 10. Microsoft Project-taulu (Microsoft Project Onlinen työpöytä-asiakasohjelmassa tehtävän taulujen käyttäminen N.d.)

Resurssienhallintaan Microsoft Projectista löytyy työkaluja erilaisista visuaalisista indikaattoreista resurssien varaamistyökaluihin ja analysointiin. Tiimin kommunikointiin Microsoft Projectista ei omaa työkalua löydy, mutta Microsoftin tuotevalikoimasta tähän löytyy esimerkiksi Skype for Busines. (Microsoft Project N.d.) Kuvio 11 on kuvattu JIRAn ja Microsoft Projectin toiminnallisuuksien eroja ja yhteneväisyyksiä.

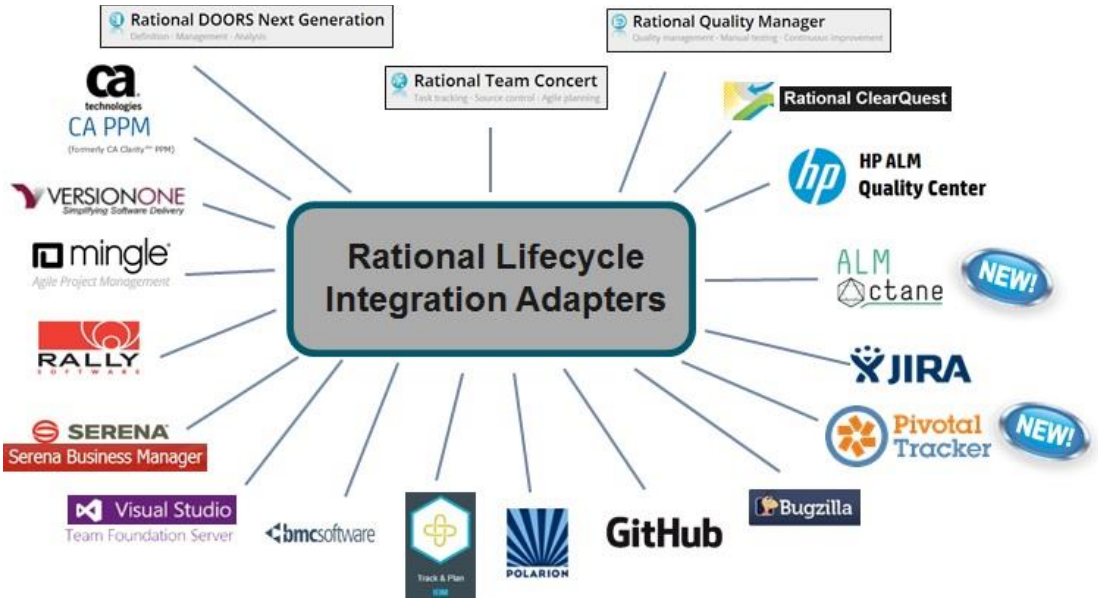


Kuvio 11. Jira vs MS-Project Comparision (JIRA Software vs Microsoft Project Comparison Chart N.d.)

4.2 IBM Rational

IBM tarjoaa ohjelmiston elinkaarenhallintaan IBM Colloborative Lifecycle Management kokonaisuutta, joka koostuu kolmesta pääkomponentista. IBM Rational DOORS Next Generation on vaatimusten- määrittelyyn ja hallintaan tarkoitettu IBM:n komponentti. IBM Rational Team Concert on suunnitteluun, työn seurantaan ja lähdekoodin hallintaan tarkoitettu komponentti. IBM Rational Quality Manager on testisuunnitelmien tekoon ja toteutukseen keskittynyt komponentti. (IBM Rational Lifecycle Integration Adapters N.d)

IBM:n työkaluihin voidaan integroida paljon eri toimittajien työkaluja tarpeen mukaan. Kuvio 12 on kuvattu erilaisia integroitavia työkaluja. Kuvio 13 kuvaa IBM Rational Team Concertin toiminnallisuuksia verrattuna JIRAn toiminnallisuuksiin.



Kuvio 12. Integration Adapters (IBM Rational Lifecycle Integration Adapters. N.d.)

Editorial information provided by Project Management Zone		
Systems	**JIRA** X	**Rational Team Concert** X
Description	Bug tracking and project planning tool, available as cloud service and as commercial tool.	Rational Team Concert is the project management component of the IBM Jazz platform.
Category	Project Planning Issue Management	Project Planning Issue Management
Project Planning Properties		
Hierarchical tasks	yes	yes
Recurring tasks	no	no
Milestone tracking	no	yes
Task dependencies	yes	yes
Critical path management	no	yes
Critical chain management	no	no
Gantt charts	yes	yes
PERT charts	no	no
Baselines	no	yes
Resource management	no	yes
Resource leveling	no	yes
Time tracking	yes	yes
Cost tracking	yes	no
Earned value management	no	no
Risk management	yes	yes
Scrum support	yes	yes
Kanban support	yes	yes
Project portfolio management	yes	yes
Issue Management Properties		
Hierarchical issues	yes	yes
Custom fields	yes	yes
Custom workflows	yes	yes

Kuvio 13. Jira and Rational Team Concert System Features

(JIRA vs Rational Team Concert System Properties Comparison N.d.)

4.3 VersionOne

VersionOne on kehittänyt erilaisia ohjelmistopaketteja ohjelmistokehityksen erilaisiin tarpeisiin. Paketit on jaoteltu siten, että erikokoiset ohjelmistoyritykset voivat ottaa omiin tarkoituksiinsa juuri sopivan version käyttöön.

Team-paketti soveltuu yhden tiimin käyttöön ja ketterien menetelmien käytön aloituspaketiksi. Ominaisuuksina tästä versiosta löytyy:

- Käyttäjätarinoiden, raportoitujen vikojen ja testien seuranta
- Toimitusten ja kehityssyklien suunnittelu
- käyttäjätarinoiden, tehtävien ja testien seurantataulut
- Läpimenoajan, tehokkuuden ja testien trendien esittäminen

(VersionOne Product Editions N.d)

Catalyst-paketti on tarkoitettu moniosaavalle tiimille ketterän kehityksen toteuttamiseen projekteissa. (VersionOne Product Editions, N.d.) Tämä paketti sisältää Team-paketin lisäksi alla listatut ominaisuudet:

- Usean projektin tuki
- Scrum ja Kanban menetelmien käytön tuki
- Sprintin toteutuksen katselmointi, Prosessin katselmointi
- Esteiden seuranta
- Kustomoitava työnkulun hallinta
- Tehostettu raportointi

(VersionOne Product Editions N.d.)

Enterprise-paketti, joka on tarkoitettu usean tiimin ketterän kehityksen työnhallintaan. Enterprise-paketissa on Catalyst-paketin sisällön lisäksi seuraavat alla mainitut ominaisuudet:

- Laajempien kokonaisuuksien hallinta
- Laajempien kokonaisuuksien hallinnan suunnittelu
- Pitkälle kehitetty raportointi
- Työnkulun metriikan esitys ja Toteutuksen onnistumisen visualisointi
- Tiimikohtaisesti muokattavat sivut eri tiimeille ohjelmistossa
- Yksilöidyt projektikohtaiset työtilat ohjelmistossa
- Projekti- ja roolipohjainen tietoturva
- Automatisoidun tuotannon ja laadunvarmistuksen visualisointi

(VersionOne Product Editions N.d.)

Ultimate-paketti on koko organisaation ketterien menetelmien toteutukseen tarkoitettu paketti, jossa on Enterprise-paketin lisäksi alla kuvatut ominaisuudet:

- Ketterä portfolion hallinta
- Portfoliotason Kanban
- tuotteensuunnittelu
- Ideoiden hallinta
- Budjetointi ja allokointi
- Regressiotestauksen hallinta
- Ohjelma- ja toteutustason visualisoinnin taulu
- SAFe tuki ja metriikka
- Monte Carlo ennusteet
- Kustomoitu raportointi

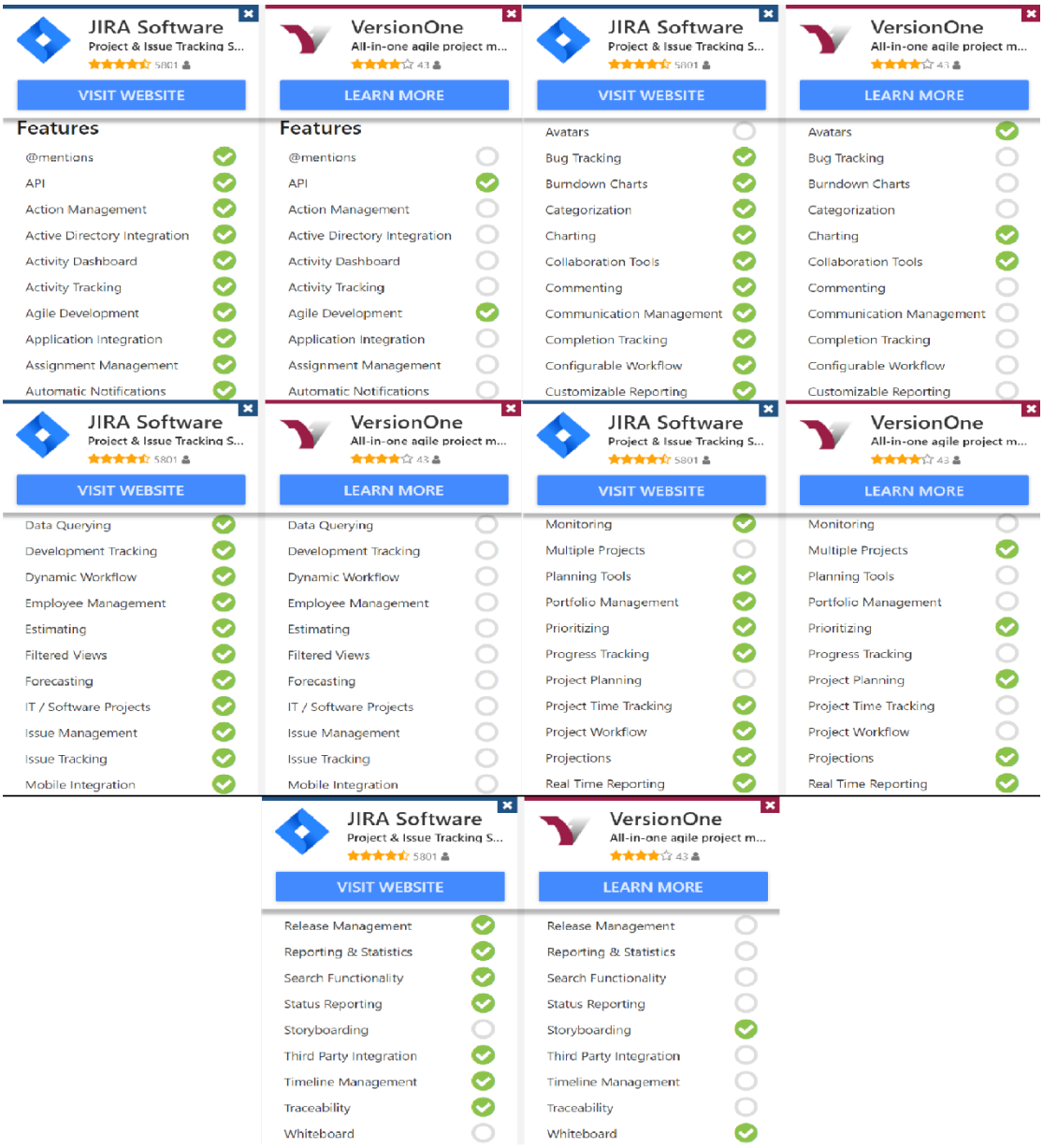
(VersionOne Product Editions N.d.)

VS-paketti on kattavin tarjolla olevista paketeista ja mahdollistaa muita VersionOne paketteja laajemman tuotantoympäristön automatisoinnin, samalla yhdistäen liiketoiminnan ketterän kehityksen toteutukseen. VS-paketti sisältää Ultimate-paketin ominaisuuksien lisäksi alle listatut toiminnallisuudet:

- Julkaisun työnkulun automatisointi
- Arvovirran hallinnointi
- Toteutuksen jäljitettävyys julkaisuista
- Toteutuksen muutoshistorian esitys aikajanalla
- Prosessin hallinta reaali-ajassa
- Automatisoidut, sekä manuaaliset aktiviteetit ja hyväksynnät
- Automatisoitu kehitysympäristö ja riskianalyysien visualisointi
- Agentiton arkkitehtuuri

(VersionOne Product Editions N.d.)

Kuvio 14 on vertailtu Version Onen ja JIRAn toiminnallisuuksia.



Kuvio 14. Jira vs VersionOne Comparison (JIRA vs Rational Team Concert System Properties Comparison N.d.)

5 Combitech Oy tuotannon työkalut

Tässä luvussa on esitelty Combitech Oy:ssä käytössä olevia tuotannon työkaluja. Combitech Oy:ssä käytetään pääsääntöisesti Atlassianin työkaluja, jotka ovat laajalti käytössä ohjelmistotuotannossa ja joihin löytyy hyvät ohjeet, sekä esimerkkejä kuinka ohjelmista saadaan eniten hyötyä erilaisilla kombinaatioilla. Näihin Atlassianin työkaluihin on saatavilla hyvin paljon erilaisia lisäosia, jotka mahdollistavat työkalujen räätälöinnin yrityksen omiin tarpeisiin, sekä ohjelmistojen vaatimusten jäljitettävyyden läpi tuotantoprosessin.

5.1 Confluence

Confluence on lähinnä dokumentaation hallintaan tarkoitettu työkalu, johon on helppo rakentaa puurakenteinen hakemistopolku ja selkeä jaottelu erityyppisten dokumenttien hallintaan. Combitech Oy:ssä tätä työkalua käytetään niin tuotannon prosessia tarkentavien dokumenttien hallintaan, kuin itse tuotteen vaatimusten hallintaan.

Vaatimuksia koskevia dokumentteja voidaan linkittää suoraan Atlassianin muihin tuotteisiin, jolloin työn valmiutta voidaan myös seurata suoraan Confluencen kautta. Näiden dokumenttipohjien, kuten Product Recuirement Template, jonka esimerkki näkyy Kuvio 18, avulla voidaan helposti käydä asiakkaan kanssa suoraan toiminnallisuuksien vaatimuksia läpi ja näitä dokumentteja voidaan päivittää prosessin edetessä.

5.2 Jira

Jira toimii tuotannon projektinhallintatyökaluna ja on linkitettynä myös koodin versionhallintaan, sekä dokumentaationhallintaan. Jira on hyvin mukautuva ja ketterään kehitykseen sopiva työkalu, johon on saatavilla erittäin paljon lisäosia niin projektinhallinnan automatisointiin, kuin testaamiseenkin. Jira:ssa on mahdollista käyttää erilaisia tauluja työn seurantaan, joka antaa tiimeille hieman liikkumavaraa tuotannon prosessissa vaikuttamatta vaatimusten jäljitettävyyteen. Tauluista on saatavilla erikseen Kanban-taulu, tai Scrum-taulu, jotka ovat luvuissa 3.2.4 ja 3.2.3 kuvattu. Myös

tauluilla tapahtuva workflow, eli tapa jolla asioita voidaan siirtää tauluilla, voidaan itse määritellä. Lisäksi taulujen sarakkeet voidaan nimetä, sekä tauluihin voidaan lisätä sarakkeita, eli niin sanottuja tiloja.

5.3 Bitbucket

Lähdekoodin versionhallintaan Combitech Oy on valinnut Atlassian:in Bitbucket ohjelman, joka mahdollistaa koodin versionhallinnan, sekä helpon linkitettävyyden yrityksessä käytettävään Jira projektinhallinta työkaluun. Bitbucket on hajautetun lähdekoodin versionhallinnan työkalu.

5.4 Jenkins

Jenkins on maailman suosituin jatkuvaan integrointiin, sekä jatkuvaan toimitukseen tarkoitettu avoimeen lähdekoodiin perustuva palvelinsovellus, joka tukee kaikkia nykyään yleisesti käytettyjä ohjelmointikieliä. (Rostedt 2018, 12-14.)

Riippumatta asennusympäristöstä tai ohjelmistoprojektien koosta, Jenkins soveltuu yritysten tarpeisiin. Jenkinsistä on saatavilla kahta eri versiota, riippuen mitä kukin Jenkinsiltä haluaa, nämä versiot ovat Weekly-, sekä LTS (Long-Term Support) -versiot. Weekly- versiota päivitetään nimensä mukaisesti viikoittain uusilla ominaisuuksilla, sekä vikakorjauksilla. Tuotannossa on suositeltava käyttää jälkimmäistä LTS-versiota, jota päivitetään harvemmin ja on näin Weekly-versiota vakaampi ympäristö. (Rostedt 2018, 12-14.)

Jenkinsiä voidaan ajaa pakettitiedostona millä tahansa käyttöjärjestelmällä, tai vaihtoehtoisesti asentaa asennuspakettien avulla. (Rostedt 2018, 12-14.)

5.5 Zephyr

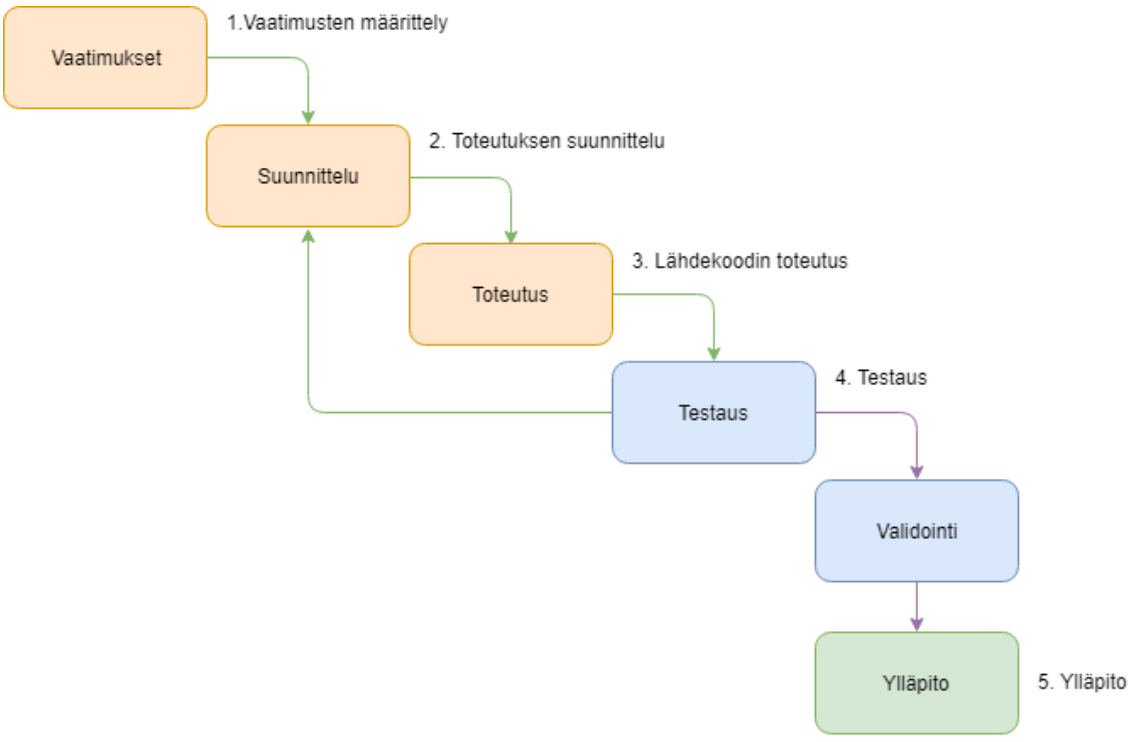
Zephyr on testauksen hallinnointiin, sekä manuaalisten testien toteutukseen tarkoitettu Jiran lisäosa. Zephyr mahdollistaa testien ja toteutuksen yhteen linkityksen, jolloin vaatimusten mukaisten testien jäljitettävyys ja raportointi on helppoa.

6 Tuotantoprosessin muutokset

Lähtötilanne:

Tässä mallissa tehtiin projektin alussa laajempi suunnitteluosuus, jossa tarjousvaiheen vaatimukset käytiin läpi ja luotiin projektille vaiheistus. Koko suunnittelumateriaali katselmoitiin läpi asiakkaan kanssa, jonka jälkeen vasta voitiin suunnitella ensimmäisen jakson toteutuksen sisältö. Prosessin vaiheet olivat noin kuuden kuukauden mittaisia ja alkoivat aina vaiheen sisällön suunnittelulla. Vaiheen loppupuolella tarkasteltiin testauksen valmiutta ja arvioitiin, tuleeko vaiheeseen suunniteltu sisältö valmiiksi kokonaisuudessaan. Vaiheen lopussa toteutusta testattiin ja katselmoitiin asiakkaan kanssa yhdestä viiteen päivään kestäneessä testaustilaisuudessa. Vaiheita projektissa oli useita ja vaatimuksia voitiin tarkentaa projektin aikana tarvittaessa.

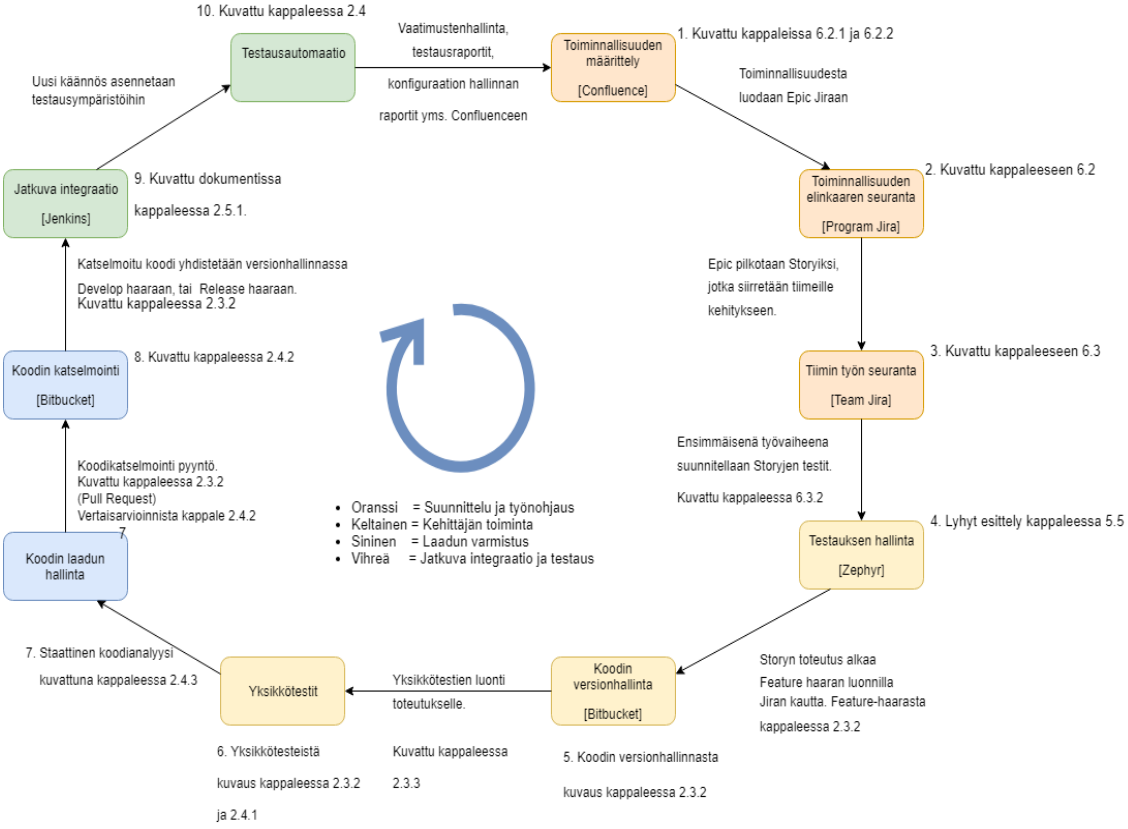
Kuvio 15 on kuvattuna ketterä- tai iteratiivinen vesiputousmalli, jossa kehitys toteutettiin pitkissä sykleissä



Kuvio 15. Ketterä vesiputousmalli

Uusi prosessi:

Koko prosessin perustana toimii SAFe ja tarkemmin Essential SAFe, joka on kuvattuna kuviossa 8. Tässä työssä toteutettu tuotannon prosessikuvaus ottaa kantaa Program-tason ohjelmistojen toteutuksen seurantaan, sekä tarkemman Team-tason toteutuksen käsittelyyn. Combitech Oy:n pelkistetty prosessin kuvaus näkyy Kuvio 16. Tuotannon yleiskuvaus



Kuvio 16. Tuotannon yleiskuvaus

Program-tasolla käsitellään uusia kehitysehdotuksia ja valmistellaan ideoita toteutukseen, sekä seurataan kokonaisten toiminnallisuuksien edistymistä toteutuksen aikana. Program-tason tarkastelua tukee Confluencen dokumentaatio, joka on linkitetty Program-tason tauluun. Confluence:n avulla voidaan käydä edistymistä läpi asiakkaan kanssa tarkastellen laajempia kokonaisuuksia.

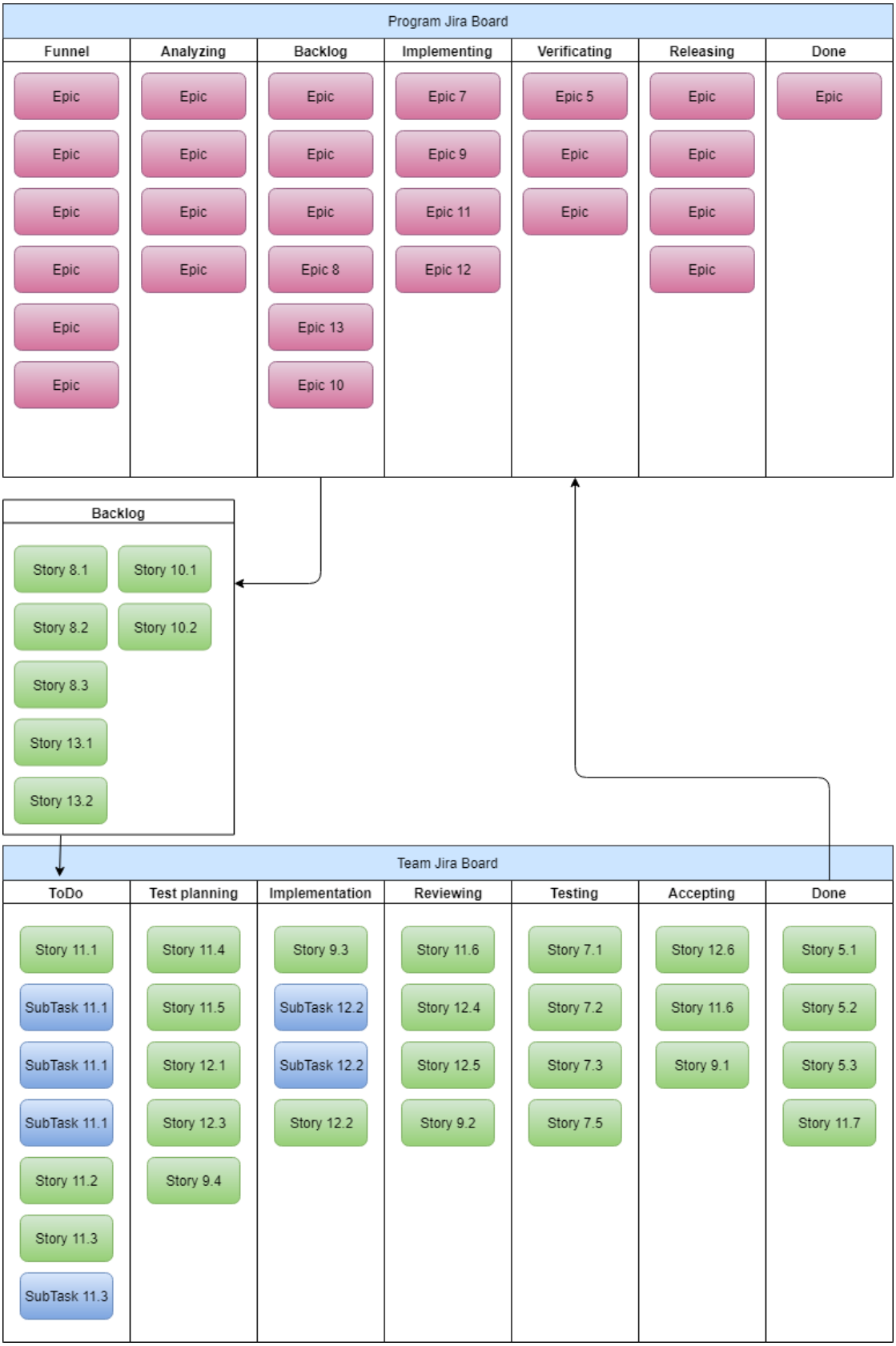
Team-tasolla seurataan itse kehitystiimien päivittäistä edistymistä ja pilkottujen ominaisuuksien kehittymistä toteutuksen aikana.

6.1 Program-taso Jira

Kuvio 17 on esitelty Program-taulun eri tilat visuaalisesti. Tällä taululla käsitellään lähinnä kokonaisia toiminnallisuuksia, eli Epic- tyyppisiä töitä. Työn alle otettavat toiminnallisuudet pyritään pitämään maksimissaan kolmen kuukauden mittaisina kokonaisuuksina, jotta nämä saadaan toteutettua yhden julkaisusyklin aikana valmiiksi.

Program-tason tauluna toimii Kanban-taulu, jolla ei ole ajallisesti määriteltyä kehitysjakson pituutta, vaan työtä toteutetaan jatkuvassa syklissä. Kaikille taulun tiloille, paitsi Funnel tilalle on määritelty WIP rajat (Work In Progress), jotta työt eivät pääse kasautumaan taululla mihinkään yksittäiseen kohtaan ja näin jarruttamaan kehityksen valmistumista. Tämä WIP raja määritellään kussakin tiimissä erikseen sopivaksi, mutta lähtöarvona voidaan pitää *n*-1, jossa *n* on kehitystyötä tekevän tiimin koko.

Jira-tauluilla sarakkeissa on kaksi tilaa, joista ensimmäiseen työ nostetaan aina edellisestä sarakkeesta tehtäväksi. Tehtävä siirretään ready tilaan sarakkeessa, kun tehtävä sarakkeessa on suoritettu. Tällä tavalla voidaan seurata kuinka kauan kukin työn osa odottaa siirtoa seuraavaan vaiheeseen ja pystytään puuttumaan mahdollisiin tuotannon ongelmakohtiin nopeasti.



Kuvio 17. Program-taulu / Tiimi-taulu

6.2 Program-taulun tilat

Program-taulun tilat ovat Funnel, Analyzing, Backlog,Implementing, Verificating, Releasing ja Done. Kappaleissa 6.2.1-6.2.7 on määritelty vaihe kerrallaan mitä toiminnallissudelle tullaan tekemään.

6.2.1 Funnel

Ensimmäinen tila Program-taululla on Funnel, jossa on Epic –tyyppiset uudet kehitysehdotukset. Tällä taulun tilalla ei ole WIP rajoitusta, koska tänne kirjataan kaikki uudet toiminnallisuudet, joita voi tulla myös kehityksen aikana. Funnel tilassa olevat Epic:it odottavat alustavaa työmäärän arviointia ja asiakkaan hyväksyntää, sekä vaatimusten tarkennusta, jotta kehitys voitaisiin aloittaa.

6.2.2 Analyzing

Seuraava tila taululla on Analyzing, jossa toiminnallisuutta käsitellään ja suunnitellaan tarkemmin, jotta saadaan riittävän tarkka kuva siitä, mitä toteutus vaatii. Lisäksi toteutukselle tehdään tässä vaiheessa työaika arvio ja toiminallisuuden hyväksyntäkriteerit määritellään asiakkaan kanssa. Hyväksyntäkriteerit kirjataan ylös Confluencesta löytyvälle Product requirement dokumenttipohjalle (Kuvio 18), johon voidaan kirjata myös asiakkaalle suunnattavia kysymyksiä toteutuksesta, sekä toteutukseen liittyviä rajauksia.

Template feature

Description

Tämä feature toteuttaa yhden toiminnallisuuden ohjelmistosta

Benefit hypothesis

toiminnallisuus antaa käyttäjille lisää oikeuksia ja nopeuttaa työntekoa

Background

Käyttäjien oikeudet eivät riitä kaikkeen tarvittavaan ja työntekeminen on liian hidasta tällä hetkellä

Assumptions

-Tätä ominaisuutta käytetään enimmäkseen puhelimella

-Toimittava Androidilla

-Toimittava myös työkoneella (Windows / Linux)

Acceptance criteria

-Toiminnallisuus täyttää järjestelmälle asetetut tietoturvavaatimukset

-Kirjautuminen ei saa kestää yli viittä sekuntia

User interaction and design

Mockup löytyy osoitteesta (URL)

Questions

-Vaikuttaako rajapinnan muutos kolmannen osapuolen komponenttiin?

Not Doing

-Ei tukea kuin windows 7, tai uudemmille windows järjestelmille

Risks

-Rajapintamuutos vaikuttaa kolmannen osapuolen komponenttiin, jolloin toimitus voi viivästyä (RISK-ID)

Linked requirements and user stories

KBAN-100 - Testi Backlog

Kuvio 18. Product Requirement template

6.2.3 Backlog

Tätä Program-taulun tilaa priorisoidaan asiakkaan kanssa jatkuvasti, jotta asiakas saa parhaan edun toteutuksesta. Toiminnallisuus siirretään Backlog- tilaan, kun sen on asiakas hyväksynyt toteutettavaksi. Tässä tilassa toiminnallisuudet priorisoidaan asiakkaan kanssa tärkeysjärjestykseen, jonka jälkeen näistä pilkotaan riittävän pieniä käyttäjätarina tyyppisiä tehtäviä tiimien toteutettavaksi. Toiminnallisuus merkitään Ready tilaan Backlog tilassa, kun kaikkiin siihen liittyviin käyttäjätarinoihin on tehty työmääräarviot ja ne on siirretty tiimien Jiraan backlog:lle odottamaan toteutusta.

6.2.4 Implementing

Neljäntenä tilana Program-taululla on Implementing, johon toiminnallisuus nostetaan, kun Team-taululle on siirretty kaikki siitä pilkotut käyttäjätarinat toteutettavaksi. Implementing tila Program-taululla sisältää kaiken, mitä luvussa 6.3 on esitelty tapahtuvan tiimin kehitystä seuraavalla taululla. Implementing tilassa toiminnallisuus merkitään valmiiksi vasta kun kaikki kyseiseen toiminnallisuuteen liitetyt Käyttäjätarinat ovat käyneet läpi Team-taulun ja päätyneet Done tilaan Team-taululla.

6.2.5 Verificating

Verificating tilassa Toiminnallisuus on integroitu jo järjestelmään testattavaksi. Confluence:n Product requirement dokumentin hyväksyntäkriteerien mukainen testaus toteutetaan tässä vaiheessa tuotantoprosessia ja toteutus esitellään osana kokonaisuutta. Toteutus merkitään valmiiksi Verificating tilassa, kun toteutus on katselmoitu osana kokonaisuutta ja hyväksytty vaatimustenmukaiseksi.

6.2.6 Releasing

Tässä vaiheessa toteutukselle tehdään julkaisun testaus, sekä päivitetään julkaisun dokumentaatio vastaamaan uutta ominaisuutta. Tämän jälkeen toiminnallisuus voidaan merkitä valmiiksi Releasing tilassa.

6.2.7 Done

Epic siirtyy Done tilaan Program-taululla, kun ominaisuus on toimitettu asiakkaalle.

6.3 Team-taulun tilat

Team-taulu on samankaltainen taulu, kuin Program-taulu mutta tällä taululla käsitellään pienempiä käyttäjätarinoita tai Sub-Task tyypin töitä eikä Epic tyypin kokonaisuuksia katso Kuvio 17. Tauluna käytetään Kanban-taulua, tai Scrum-taulua, riippuen siitä kumpaa taulua projektitiimi haluaa käyttää.

Tiimien taulut sisältävät seitsemän eri tilaa, jotka kuvastavat Käyttäjätarinan, tai Sub-Taskin valmiustilaa sprintin sisällä. Tilat ovat ToDo, Test planning, Implementation, Reviewing, Testing, Accepting ja Done. Taulun eri tiloille on tehty valmiin määritelmät, jotka löytyvät Kuvio 19.

6.3.1 ToDo

Tähän tilaan nostetaan tiimin taululle Käyttäjätarinat toteutusta varten Backlogilta. Scrum-taululle jokainen tiimi ottaa ToDo tilaan seuraavan kehityssyklin työt työmääräarvioiden mukaan toteutettavaksi. Kanban-taulua käytettäessä voidaan työtä tehdä ilman tarkkaa aikaan rajattua jaksotusta, mutta Kanban-taululla tärkeää on WIP-limitit, eli töiden rajoitukset kussakin taulun tilassa, jotta työt eivät ruuhkaudu taululla johonkin tiettyyn tilaan. Tässä tilassa kehittäjä voi vielä tarvittaessa pilkkoa Käyttäjätarinaa pienemmiksi tehtäviksi Team-taululle. Kuvausta tarkennetaan riittävälle tarkkuudelle, jotta testien suunnittelu voidaan aloittaa, jos tätä ei ole aiemmin tehty.

6.3.2 Test planning

Toinen vaihe on testien suunnitteluvaihe, jossa suunnitellaan toiminnalliset ja ei-toiminnalliset testit, jotka todentavat työn alle otetun toteutuksen valmiutta. Ei-toiminnallisista testeistä lisää luvussa 2.4.7. Tässä vaiheessa testeistä tehdään sanalliset selkeät kuvaukset, joiden pohjalta testejä on helppo työstää myöhemmin lisää. Tähän vaiheeseen osallistuvat kehittäjät, testaajat ja tarvittaessa projektipäällikkö. Kun testit on suunniteltu, ne merkitään valmiiksi odottamaan toteuttamista.

6.3.3 Implementation

Tämä on tuotantoprosessin se vaihe, jossa luodaan itse lähdekoodi, sekä yksikkötestit toteutukselle. Yksikkötestit ovat tärkeä osa toteutusta, koska näitä voidaan ajaa automaattisesti ja käyttää esimerkiksi regressiotestaukseen kehityksen aikana.

Ohjelmiston dokumentaatio päivitetään Implementation tilassa vastaamaan uutta toteutusta. Kehittäjä tarkistaa projektin DoD (Definition Of Done) mukaiset asiat toteutuksesta, jonka jälkeen toteutus voidaan merkitä Team-taulun Implementation tilassa valmiiksi. Definition Of Done on kuvattu Kuvio 19.

6.3.4 Reviewing

Reviewing tilassa toteutuksen katselmoi tiimin toinen kehittäjä, jolla on riittävä näkemys ohjelmistokokonaisuudesta. Tällä varmistetaan koodin laadulliset vaatimukset, sekä ehkäistään selkeiden virheiden jääminen toteutukseen. Kun koodi on katselmoitu, se merkitään Reviewing tilassa valmiiksi odottamaan seuraavaa vaihetta. Katselmoinnin tekee aina joku muu, kuin itse toteutuksen kehittäjä.

6.3.5 Testing

Test planning tilassa aloitettu testien suunnittelu tehdään tässä vaiheessa prosessia loppuun, jonka jälkeen nämä testit ajetaan testiympäristössä ja varmistetaan, että kaikki testit menevät läpi. Jos toteutuksesta löytyy virheitä, ne raportoidaan ja osoitetaan tiimille korjattavaksi.

Tässä tilassa tarkastetaan niin toiminnallisten-, kuin ei-toiminnallisten testien läpimeno, ennen kuin toteutus voidaan merkata valmiiksi Testing tilassa. Automaatiotestien luomisesta vastaa automaatiotestaaja kehittäjien avustuksella. Automaatiotesteistä lisää luvussa 2.4-2.4.7.

6.3.6 Accepting

Team-taulun Accepting tilassa projektipäällikkö ja ohjelmistoarkkitehti katselmoivat toteutuksen ja sitä koskevan dokumentaation ja hyväksyy toteutuksen valmiiksi Team-taululla. Mikäli asiakkaan edustaja ei pääse paikalle toteutusta hyväksymään, voi toimittajan projektipäällikkö hyväksyä toteutuksen.

6.3.7 Done

Tässä vaiheessa valmiit Käyttäjätarinat odottavat, että kaikki toisiinsa liittyvät käyttäjätarinat käyvät Team-taulun vaiheet läpi, jonka jälkeen Epic, johon nämä Käyttäjätarinat liittyvät jatkaa tuotantoprosessin kulkua Program-taululla.

6.4 Valmiin määritelmä

Combitech Oy:ssä on tuotannon prosessin tiimitason taulun vaiheille määritelty omat valmiin määritelmänsä, joten kehittäjä voi tarkistaa tämän määritelmän mukaan työnsä, ennen kuin merkkaa työn valmiiksi tilassa. Kuvio 19 kertoo eri tilojen valmiin määritelmän.

ToDo	Test planning	Implementation	Reviewing	Testing	Accepting	Done
* Story on pilkottu riittävän pieniksi Sub-Taskeiksi toteutettavaksi * Kuvaus tarkennettu riittävän tarkaksi, jotta testauksen suunnittelu voidaan aloittaa	* Storyyn on liitetty testitapaus * Bug / Vika on kuvattu niin tarkasti, että testaus onnistuu. Usein asiakkaalta tulevia kuvauksia on syytä tarkentaa, jotta testaus ymmärtää mistä on kyse.	* Koodi versionhallinnassa * Koodi tyyliohjeen mukaista * Koodi kääntyy Jenkinsillä * Yksikkötestit tehty toteutetulle osuudelle, tai korjattu muuntuneelle osuudelle * Kaikki yksikkötestit menee läpi * Dokumentaatio päivitetty	* Koodi on testattu vartaisarvioijan toimesta kehitysympäristössä * Koodi on katselmoitu silmämääräisesti läpi vertaisarvioijan toimesta	* Toiminnalliset testit on viimeistelty ja menee läpi testausympäristössä * Ei-toiminnalliset testit tehty ja menee läpi	* Toiminnallisuuden, testiraportit ja dokumentaatio on käyty läpi projektipäällikön toimesta * Projektipäällikkö on hyväksynyt käyttäjätarinan	* Käyttäjätarina on viety testausympäristöön odottamaan programtason hyväksyntää

Kuvio 19. Valmiin määritelmä

6.5 Testaus

Testaus Combitech Oy:ssä kattaa ohjelmakoodin koodianalyysin, yksikkötestauksen, vertaisarvioinnin, integraatiotestauksen ja toimituksen testauksen, sekä tutkivan testauksen. Koodianalyysi, yksikkötestaus, vertaisarviointi ja integraatiotestaus on selitetty kappaleessa 2.4

6.5.1 Toimituksen testaus

Toimitukseen menevä ohjelmistoversio asennetaan testausympäristöön ja samalla testataan ohjelmiston asennuspakettien toimivuus. Testausta toteuttavalla tiiminjäsenellä täytyy olla selkeä käsitys loppukäyttäjän ohjelmiston käyttötavoista, sekä ohjelmiston vaatimuksista, jotta hän voi toteuttaa toimituksen testausta. Ohjelmiston toteutusta koskevat vaatimukset löytyvät Confluencen Product Requirement template dokumentista.

Osana toimituksen testausta ohjelmisto asennetaan tuotantoympäristöön ja aiemmin suunnitellut testitapaukset ajetaan läpi tuotantoympäristössä. Tässä testataan ohjelmiston vaatimustenmukaisuutta, eli suoritetaan validointitestaus. Validointitestausta kutsutaan myös SAT-testiksi eli Site Acceptance Test.

6.5.2 Tutkiva testaus

Tutkiva testaus on ohjelmiston käyttöä ja havainnointia käytön aikana. Testaaja valitsee komponentin, johon tutkivassa testauksessa keskittyy ja alkaa käyttää komponenttia mahdollisimman monipuolisesti. Tämä tutkiva testaus paljastaa usein ohjelmistovirheitä, jotka jäisivät muuten automatisoiduilta testeiltä tai suunniteltuja testitapauksia suorittaessa huomaamatta.

6.6 Lähdekoodin versionhallinta Combitech Oy:ssä

Combitech Oy:ssä lähdekoodin versionhallintaan käytetään Atlassianin Bitbucket:ia, mutta itse Git työkalu ei ole merkityksellinen tuotantoprosessin kannalta, vaan tärkeintä on yhdenmukaiset käytänteet koodin versionhallinnassa.

Combitech Oy:ssä oli käytössä Git-versionhallinta, mutta eri projektit käyttivät gittiä kukin omalla haluamallaan tavalla, jolloin myös lähdekoodin versionhallinnan käytänteet täytyi opetella, kun kehittäjä siirtyi projektilta toiselle.

Uuden prosessin mukana lähdekoodin versionhallinta pyritään ainakin pääpiirteittäin yhdenmukaistamaan Gitflow workflown mukaiseksi, jolloin projektikohtaisen perehtymisen tarpeen tulisi vähentyä. Gitflow workflow on kuvattu aijemmin kappaleessa 2.3.2.

6.7 Käännösympäristö

Käännösympäristön tekniseen toteutukseen, kuten siihen käytetäänkö ympäristössä hyödyksi konttiteknologiaa tai virtuaalisia ympäristöjä ei tässä opinnäytetyössä oteta kantaa. Käännösympäristössä käytettävät perustyökalut pyritään pitämään kuitenkin yrityksen sisällä samoina, kuten myös työkalujen käyttötavat.

Tuotannon käännösten ja testauksen automatisoijana toimii Combitech Oy:ssä Jenkins, joka on erittäin monipuolinen ja tehokas web-pohjainen palvelinsovellus. Lähdekoodista tuotetaan automatisoidusti uusia ohjelmiston versioita päivittäin. Näihin versiopäivityksiin on sidottu myös automatisoituja testejä ajettavaksi, jolloin uudet versiot on testattuja valmistuessaan. Jatkuva integraatio tilaajan ympäristöstä löytyy, mutta jatkuvaa toimitusta varten automaattisia testejä on vielä kehitettävä lisää. Testauksesta ja sen tuloksista on enemmän luvuissa 2.4 ja 6.5.

Jatkuva integraatio ja toimitus Combitech Oy:ssä:

Combitech Oy:ssä jatkuvaa integraatiota toteutetaan Jenkinsin tehtävien avulla. Pääsääntöisesti ohjelmistot ovat suuria ja näihin on luotu automatisoituja testejä laadunvarmistusta varten.

Laajojen ohjelmistokokonaisuuksien kääntäminen ja automaattisten testien ajaminen voi viedä useita tunteja ja samalla Jenkinssin käännösympäristöllä voi olla useampiakin käännöstöitä allokoituna, joten nämä automatisoidut integraatiot on ajastettu tapahtumaan yöllä, jolloin laajojenkaan kokonaisuuksien kääntäminen ei haittaa kehitystyötä. Tarvittaessa Jenkinsistä voidaan myös käynnistää nämä automatisoidut koostamiset manuaalisesti, jolloin ohjelmiston korjauksia ja toteutusta voidaan testata nopeasti.

Yksi Combitech Oy:n keskeisimmistä tuotantoprosessin kehityskohteista lähitulevaisuudessa tulee olemaan jatkuvan toimituksen kehittäminen.

7 Pohdinta

Tämän työn tavoitteina oli dokumentoida yhdenmukaiset toimintatavat ja puitteet ohjelmistokehitykselle ja näin tehostaa työhön perehtymistä, sekä tuotantoprosessia. Tällä hetkellä prosessikuvaus löytyy dokumentoituna Confluencesta ja on käytössä osassa tilaajan projekteissa. Lähtökohtana kehitykselle oli ketterät kehitysmallit, sekä korkean tason prosessit, jotka määrittelevät ohjelmistotuotannon vaadittavat hallinnolliset menettelyt. Tuotantoprosessi mukailee SAFe viitekehystä, joka antaa hyviä esimerkkejä työn Program-tason kuvaamisesta sekä tiimin päivittäisen työnkulun edistymisen kuvaamisesta. Työkaluiksi oli valikoitunut Atlassianin tuotteet, jotka ovat olleet jo pitkään laajalti käytössä ohjelmistokehityksessä, joten kaikkeen prosessissa tarvittavaan löytyy helposti integroitavia lisäosia. Lisäksi Atlassianin tuotteisiin löytyy paljon hyviä käytännöllisiä esimerkkejä, sekä ohjeita tuotteiden optimaaliseen käyttöön.

Tarve käyttää samoja henkilöresursseja useammassa projektissa aiheuttaa ylimääräistä työtä, kun tuotantoprosessit ja työkalut vaihtelivat projekteittain. Yhtenäinen tuotantoprosessi ja työkalut vähentävät myös ICT-tuen tarvetta, kun tuotannon työkalujen konfigurointi toteutetaan saman pohjan mukaisesti. Tässä työssä kuvattu tuotantoprosessimalli on yleiskuvaus, jota täydentää projektien omat dokumentoidut yksityiskohdat prosessissa. Täysin vakioitua yhtä oikeaa mallia prosessille on mahdoton kuvata, koska eri projektien tuotannot voivat olla hyvin erityylisiä, jolloin pienet muutokset toimintatavoissa ovat välttämättömiä. Perusprosessi on kuitenkin aina kuvauksen mukainen, joten uusien työntekijöiden ei tarvitse opetella projektista toiseen siirtyessään täysin uusia työskentelytapoja.

Tuotantoprosessin yhdenmukaistaminen läpi yrityksen on aikaa vievä ja haasteellinen prosessi, varsinkin kun eri yksiköissä on totuttu erilaisiin toimintatapoihin ja jopa eri yksiköiden sisällä eri projektitiimit käyttävät erilaisia toimintatapoja. Tämän työn tuloksena toteutettu prosessikuvaus on pohja, jota aletaan viemään käytäntöön pienissä erissä, jotta nopea ja radikaali muutos prosessissa ei hidastaisi kehitystä, tai pahimmassa tapauksessa pysäyttäisi kehitystä hetkellisesti kokonaan. Kun prosessi on saatu käyttöön kokonaisvaltaisesti, aletaan kerätä käyttäjäkokemuksia kehittäjiltä, sekä asiakkailta. Näiden kokemusten pohjalta prosessia kehitetään eteenpäin, joten

prosessi tulee kehittymään jatkuvasti vastaten ohjelmistoalan muutoksien tarpeita myös tulevaisuudessa.

Confluencen tarjoamat dokumenttipohjat mahdollistavat selkeän ja ajantasaisen raportoinnin kehityksestä ja kehitysideoista asiakkaalle sekä kehitystiimille, jolloin käytössä olevien nykyisten dokumenttien manuaalinen käsittely vähenee huomattavasti, sekä reagoiminen asiakastarpeisiin nopeutuu. Seuraava askel prosessin kehityksessä onkin Confluencessa olevan dokumenttipohjan rakenteen vakioiminen, jolloin jokaisen projektin dokumentaatio, kuten esimerkiksi vaatimukset tai arkkitehtuurikuvaukset on jokaisella projektilla samassa kohtaa puurakennetta. Tällä pyritään helpottamaan henkilöresurssien käyttöä yli projektien entisestään.

Hyvin suunniteltu ja toteutettu työnkulun seuranta Jirassa antaa selkeän kuvan kehityksen tilanteesta ja mahdollistaa myös kehityksen Program-tason tarkastelun asiakastapaamisien yhteydessä. Projektin hallinto voi tämän prosessin mukaisen toteutuksen avulla tarkastella nopeasti niin julkaisua koskevaa kehityksen kokonaistilannetta, kuin tiimin päivittäistä kehityksen kulkua, jolloin prosessin ongelmakohdat voidaan havaita ja prosessia kehittää näiltä osin eteenpäin.

Yksi iso kehityskohde työssä oli lähdekoodin versionhallinnan, eli Git:in käytön yhdenmukaistaminen. Ennen prosessikuvausta jokainen kehitystiimi oli luonut omat versionhallinta-haaransa ja käyttötavat vaihtelivat täysin eri projekteissa. Lähdekoodin versionhallinnan ymmärtäminen uusille tiimin jäsenille vei useamman päivän, joskus pidempäänkin ja inhimillisten virheiden riski oli pitkään suuri. Yhdenmukaisten Gitin käytön käytänteiden jälkeen tulevaisuudessa kehittäjän ei tarvitse enää opetella aina uusia tapoja lähdekoodin hallinnasta vaihtaessaan projektia, tai tehdessään töitä useammassa projektissa yhtä aikaa. Tässä työssä määritelty lähdekoodin versionhallinta perustuu yleisesti käytössä olevaan Gitflow malliin, mutta kuvauksessa on pyritty avaamaan tarkemmin useamman ohjelmistoversion yhtäaikaisen toimituksen vaatimia versionhallinnan käytänteitä.

7.1 Tunnistetut mahdollisuudet

Yhtenäisen prosessin kehittäminen ja arvioiminen on kattavampaa, kuin yksittäisten projektikohtaisten prosessien kehittäminen.

Toteutuksen seuranta helpottuu, kun vakioitujen työkalujen käyttötavat ja mahdollisuudet tulevat tutuiksi koko organisaatiossa, samalla saadaan laajempi käyttäjäkunta antamaan kehitysehdotuksia prosessin eri vaiheisiin.

Kun työn alle otettavat tehtävät suunnitellaan hyvin ja määritellään tarkemmin juuri ennen toteutuksen aloittamista, odottamattomien ongelmien ja turhan työn määrä vähenee kehitystyössä. Tätä suunnittelutyötä tukee esimerkiksi Confluencen valmiit dokumenttipohjat.

Kun työt määritellään ja suunnitellaan vain siltä osin kun toteutusta tullaan seuraavaan toimitukseen tekemään, ei turhaa määrittelytyötä projektissa tehdä, eikä työlistalla ole vuosia sitten määriteltyjä vanhentuneita työtehtäviä.

7.2 Tunnistetut tulevaisuuden haasteet

Tuotannon muuttaminen enemmän testausta tukevaksi tulee aiheuttamaan totuttelua tiimien sisällä ja tämä muutos tulee viemään aikaa, jotta uudet käytänteet tulevat osaksi kehityksen rutiineja.

Jatkuvaa suunnittelua vaativa lähestymistapa vaatii paljon aikaa ja totuttelua niin yrityksen sisällä, kuin asiakkaan puolella, mutta tulee varmasti lisäämään toteutuksen arvoa asiakkaalle ja todellisuudessa nopeuttamaan kehitystä, kun toteutus on entistä paremmin ja oikeaan aikaan suunniteltua.

Jatkuva suunnittelu on ketterää kehitystä parhaimmillaan, mutta saattaa olla vaikea toteuttaa puolustusteollisuuden alalla, koska budjettitalous ja kilpailutus vaativat melko tarkkaa ennalta suunnittelua varsinkin kiinteähintaisia tarjouksia laadittaessa.

7.3 Suositeltuja toimenpiteitä

Prosessi on otettava vaiheittain pala kerrallaan käyttöön ja varmistettava, että prosessia ryhdytään noudattamaan tarkasti, jotta jatkossa voidaan arvioida prosessin kehityskohteita ja hyviä puolia.

Tulevaisuudessa tuotantotapojen kehityskeskustelut olisi hyvä ottaa osaksi tiimien jaksottaisia palavereita ja kannustaa kaikkia tiimin jäseniä omalta osaltaan arvioimaan tuotantoprosessin hyviä ja huonoja puolia.

Prosessin muutoksista on syytä puhua avoimesti heti kun muutokset otetaan käyttöön, mutta prosessia muuttaviin toimenpiteisiin ei tulisi ryhtyä, ennen kuin prosessin käytänteisiin on tullut riittävät rutiinit. Muutosvastaisuus on normaali ilmiö ja se pitää hyväksyä.

Kehitystä tiimeissä olisi hyvä tehdä Kanban, tai Scrumban menetelmillä ennemmin, kuin pyrkiä puhtaasti Scrum tyyppiseen menetelmään. Tällöin mitattaisiin läpimenoaikaa, eikä valmistuneen työn määrää, jolloin tiimien koon vaihtelevuus ja henkilöstön käyttö yli projektien ei vaikuttaisi niin paljon kehityksen seurantaan.

Lähteet

Aaronen. 2017. Resurssienhallinta-ohjelmiston käyttäjäkeskeinen suunnittelu. Opinnäytetyö. Turun ammattikorkeakoulu. Viitattu 23.3.2019. http://urn.fi/URN:NBN:fi:amk-2017121521510.

Cunningham. 2001. Manifesto for Agile Software Development. Verkkosivut. Viitattu 21.3.2019. https://agilemanifesto.org/.

DevOps. N.d. Verkkosivut. Viitattu 3.4.2019. https://www2.wakaru.fi/devops/.

Essential SAFe. 2017. Verkkosivut. Viitattu 23.7.2018. https://www.scaledagileframework.com/essential-safe/.

Gitflow workflow. N.d. Verkkosivut. Viitattu 26.7.2018. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

Haikala, I & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Hämeenlinna: Kariston Kirjapaino Oy.

Huttunen, J. 2017. Tehokas siirtyminen järjestelmän kehityksestä ylläpitoon. Opinnäytetyö. Jyväskylän ammattikorkeakoulu. Viitattu 1.8.2018. http://urn.fi/URN:NBN:fi:amk-2017121821764.

IBM Rational Lifecycle Integration Adapters – Tasktop Edition. N.d. Verkkosivut Viitattu 1.12.2018. https://jazz.net/products/rational-adapters-tasktop-edition.

JIRA Software vs Microsoft Project Comparison Chart. N.d. Verkkosivut. Viitattu 23.3.2019. https://www.getapp.com/project-management-planning-software/a/jira/compare/microsoft-project/.

JIRA vs Rational Team Concert System Properties Comparison. N.d. Verkkosivut. Viitattu 22.3.2019. https://project-management.zone/system/jira,rational-team-concert.

JIRA Software vs VersionOne Comparison Chart. N.d. Verkkosivut. Viitattu 23.3.2019. https://www.getapp.com/project-management-planning-software/a/jira/compare/versionone/.

Layton, M. C. 2015. Scrum For Dummies. Wiley. Viitattu 23.7.2018. http://library.books24x7.com.ezproxy.jamk.fi:2048/toc.aspx?bookid=82533.

Learn Version Control with Git, N.d, Viitattu 15.12.2018, https://www.git-tower.com/learn/git/ebook/en/desktop-gui/appendix/from-subversion-to-git.

Leopold, K & Kaltenecker, S. 2015. Kanban Change Leadership. Wiley & Sons. Viitattu 1.8.2018. https://janet.finna.fi/Record/nelli14.3710000000371916.

Lähteinen, M. 2015. Ohjelmistokehitysprosessin laadun ja kustannustehokkuuden kehittäminen. Ohjelmistotekniikan diplomityö. Vaasan yliopisto. https://www.tritonia.fi/fi/e-opinnaytteet/tiivistelma/6426/Ohjelmistokehitysprosessin+laadun+ja+kustannustehokkuuden+kehitt%C3%A4minen.

Majchrzak, M. 2017. Experience Report: Introducing Kanban Into Automotive Software Project. Viitattu 1.8.2018. https://janet.fi.

Microsoft Project. N.d. Verkkosivut. Viitattu 24.3.2019. https://products.office.com/en-US/project/project-management.

Microsoft Project Onlinen työpöytä-asiakasohjelmassa tehtävän taulujen käyttäminen. Nd. Verkkosivut. Viitattu 24.3.2019. https://support.office.com/fi-fi/article/microsoft-project-onlinen-ty%C3%B6p%C3%B6yt%C3%A4-asiakasohjelmassa-teht%C3%A4v%C3%A4n-taulujen-k%C3%A4ytt%C3%A4minen-1b9b44d7-fd8e-4b3b-ab94-2b97deb9945b?ui=fi-FI&rs=fi-FI&ad=FI#bkmk_makeyourprojectagile.

Ohjelmistotuotanto. N.d. Verkkosivut. Viitattu 1.3.2019. https://qmea.fi/ohjelmistot.php.

Pahuja S. N.d. What is Scrumban?. Blogi. Viitattu 20.8.2018. https://www.agilealliance.org/what-is-scrumban/.

Penttilä, E. 2014. C++-ohjelmien laadun parantaminen staattisella koodianalyysillä. Gradu. Aalto University. Viitattu 3.8.2018. http://urn.fi/URN:NBN:fi:aalto-201406252203.

Project management zone. nd. JIRA vs Rational Team Concert System Properties Comparison. Verkkosivu. Viitattu 14.3.2019. https://project-management.zone/system/jira,rational-team-concert.

Rostedt, J. 2018. Ohjelmistotyön tehostaminen automaatioympäristössä Case: Jenkins. Opinnäytetyö. Satakunnan ammattikorkeakoulu. Viitattu 1.8.2018. http://urn.fi/URN:NBN:fi:amk-201805015983.

Tersa, T. 2002. Testausmenetelmien käyttö sovelluksen systeemitestauksessa. Tietotekniikan pro gradu. Jyväskylän yliopisto. Viitattu 22.11.2018. http://www.mit.jyu.fi/opetus/opinnayte/gradu/systeemitestaus/Tiina_Tersa.pdf.

The Ultimate Agile Guide. N.d. Eylean. The-Ultimate-Agile-Guide.pdf. Viitattu 6.8.2018. https://www.eylean.com/blog/2016/08/the-ultimate-agile-guide/.

The Scrum Guide. 2017. Versio 11/2017. Viitattu 12.9.2018. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100.

Tyynismaa,M. 2014. Lean- ja Kanban-menetelmät ohjelmistotuotannossa. Opinnäytetyö. Seinäjoen ammattikorkeakoulu, Viitattu 9.12.2018. https://www.theseus.fi/bitstream/handle/10024/78004/Mika_Tyynismaa.pdf?sequence=1.

Vacanti, D. 2018. The Kanban Guide for Scrum Teams. Scrum.org. Viitattu 20.8.2018. https://www.scrum.org/resources/kanban-guide-scrum-teams.

VersionOne Product Editions. N.d. Verkkosivut. Viitattu 9.12.2018. https://www.collab.net/products/versionone-editions.

Viisi ksymystä, N.d. Verkkosivut. Viitattu 15.12.2018. http://www.sixsigma.fi/fi/lean/yleinen/viisi-kysymystae/.