Tampereen yliopisto

Jethro Lehtojärvi

DEVOPS TOIMINTAMALLI JA TESTAUSAUTOMAATIO VALIDOINNIN TYÖKALUINA

Lääketeollisessa toimintaympäristössä.

Johtamisen ja talouden tiedekunta
Prof. Kari Systä
Prof. Hannu-Matti Järvinen
Diplomityö
Marraskuu 2023

TIIVISTELMÄ

Jethro Lehtojärvi : DevOps toimintamalli ja testausautomaatio validoinnin työkaluina
Tampereen yliopisto
Johtamisen ja talouden tiedekunta
Diplomityö
Marraskuu 2023

Teollisuuden neljäs vallankumous on käynnissä, joka kannustaa digitaaliseen murrokseen, jossa tekoälyllä ja muilla nousevilla teknologioilla pyritään edistämään liiketoimintaetuja. Tämän asetelman keskeinen haaste on jatkuvat liiketoimintatarpeiden muutokset ja datan käsittelyn ja hyödyntämisen räjähdysmäinen kasvu. Lääketeollisuus ei tee tässä poikkeusta, mutta lääkealan perinteinen, voimakkaasti riskiohjautuva spesifikaatioverifikaatioelinkaarimalli validointitoiminnassa on haasteellinen nopeasti muuttuvassa liiketoimintakentässä, koska sen toimintaperiaatteisiin kuuluu tarkka vaatimuksien määrittely ennalta.

Ketterä elinkaarimalli tarjoaa vaihtoehdon, joilla tähdätään siihen, että uusia vaatimuksia toteuttaa nopeasti ja usein, eikä lopullisen ratkaisun kaikkia vaatimuksia tarvitse tietää alussa, ne opitaan tekemisen aikana. Ketterissä toimintamalleissa on pohjimmiltaan kyse kyvystä luoda uusia ominaisuuksia eli pyrkiä jatkuvaan muutokseen, joka lääketeollisuuden hyvien tuotantotapojen hallinnan kannalta vaikuttaa melkoiselta haasteelta.

Tämän tutkimuksen ensisijaisena tavoitteena oli tutkia ketterän elinkaaritoimintamallin toimintaedellytyksiä lääkeliiketoiminnan perinteisen spesifikaatioverifikaatioelinkaarimallin sijasta ja toisarvoisena tutkia testausautomaation edellytyksiä osana ohjelmistotestauksen validointiprosessia. Teoreettisen viitekehyksen lisäksi työn tavoitteena oli testata havaintoihin perustuen, miten Azure DevOps työkaluohjelmistoa voitaisiin soveltaa järjestelmäintegraatiosovelluksen elinkaaren hallinnan välineenä hyvien tuotantotapojen kriteereitä täyttäen.

Tutkimus toteutettiin perehtymällä lääketeollisuuden toimintaa ohjaaviin erilaisiin viranomaisohjeisiin, sekä ISPE:n GAMP 5 julkaisuihin. Tämä lisäksi perehdyttiin sovellusteollisuuden käyttämään ketterään toimintamalliin ja niihin liittyviin testausautomaatiokäsitteisiin ja testaus- ja toimitusputkiin, jotka kaikki yhdessä muodostavat käsitteen DevOps. Edellä mainittujen lisäksi, tilaajan toimintaan tutustuttiin keskustelemalla eri asiantuntijoiden kanssa.

Tutkimuksen teknisessä osassa sovelletaan Azure DevOps työkaluohjelmiston kyvykkyyksiä toteuttaa hyvien tuotantotapojen mukaisen elinkaarimallin toimenpiteitä DevOps toimintamallien mukaisesti järjestelmäintegraation validoimiseksi, eli pyritään osoittamaan sen soveltumisesta aiottuun lääkeliiketoiminnalliseen käyttöön. Työ esittää esimerkkien omaisesti validointityötä tukevan DevOps toimintamallin, joka sisältää vaatimusmäärittelyn, testausautomaation, riskienhallinnan, sekä testaus- ja toimitusputket. Lisäksi paneudutaan vaatimuksiin käyttäjävaatimuksien jäljityksen ja versiohallinnan osalta, sekä niiden toteutumiseen. Ratkaisu ei täytä kaikkia hyvien tuotantotapojen mukaisia elinkaarenhallinnan toimenpiteitä, mutta mahdollistaa ketterän sovelluskehityksen lääkeliiketoimintaa harjoittavissa yrityksissä, kunhan sitä täydennetään joidenkin puutteiden osalta muilla ratkaisuilla. Kuten esimerkiksi, vaatimukset koskien validointiraportin liitteitä ja käyttäjähyväksyntätestien suoritusraportteja.

Avainsanat: järjestelmäintegraatio, säädelty liiketoiminta, validointi, ohjelmistotestaus, testausautomaatio, GMP, GxP, DevOps, CI/CD.

ABSTRACT

Jethro Lehtojärvi : DevOps operating model and test automation as validation tools
Tampere University
Faculty of Management and Business
Master's Thesis
November 2023

The fourth industrial revolution is underway, spurring a digital revolution where artificial intelligence and other emerging technologies aim to advance business benefits. The crucial challenge of this layout is the continuous changes in business needs and the explosive growth of data processing and utilization. The pharmaceutical industry is no exception to this, but the pharmaceutical industry's traditional, heavily risk-driven specification verification lifecycle model in system validation is challenging in a rapidly changing business field, because its operating principles include the precise definition of requirements in advance.

Agile life cycle model offers an alternative that aims to implement new requirements quickly and often, and there is no need to know all the requirements of the final solution at the beginning, these are learned while progressing. Agile operating model is basically about the ability to create new features, i.e. strive for continuous change, which may seem quite challenging setup to be managed in the pharmaceutical industry in context of good manufacturing practice.

The primary aim of this thesis was to study the operating capabilities of an agile life cycle model in the pharmaceutical business instead of the specification verification model. The secondary aim was to understand what kind of conditions are needed for use of test automation as part of the pharmaceutical software validation process. In addition to the theoretical references, there was goal to test these theoretical findings with the Azure DevOps tool as a life cycle management tool for system integration solution while meeting good manufacturing practice criteria.

The study was carried out by familiarizing ourselves with the official guidelines governing the pharmaceutical industry practice, as well as ISPE's GAMP 5 publications. In addition to this, an agile operating model was studied including test automation concepts and testing and delivery pipelines, which together form the concept of DevOps. To fulfill our understanding there were discussions with customers' subject matter experts as well.

In the technical part of this study, the capabilities of the Azure DevOps tool are applied to implement good manufacturing practice compliant DevOps life cycle model for system integration solution, i.e. the demonstrate applicability to the intended pharmaceutical business use. The implementation covers DevOps operating model that supports the validation work, which includes requirements definition, test automation, risk management and testing and delivery pipelines. In addition, attention will be paid also to the requirements concerning user requirement tracking and version control. The solution does not fulfill all measures of life cycle management according to good manufacturing practice. But it enables agile application development in companies engaged in pharmaceutical business, when it is supplemented with other solutions for some shortcomings, such as, the requirements regarding the preparation of appendices for validation report and user acceptance testing execution reports.

Keywords: system integration, regulated business, validation, software testing, test automation, GMP, GxP, DevOps, CI/CD.

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

CD	Continuous Delivery
CDE	Continuous Deployment
CFR	Code of Federal Regulations
CI	Continuous Integration
CI/CD	ks. CI ja CD
CMMI	Capability Maturity Model Integration
DevOps	Development and Operations (Kehitys ja operaatiot)
EMA	European Medicines Agency, Euroopan lääkealan turvallisuus- ja kehittämiskeskus.
EU Annex 11	Tarkoitetaan EU:n säätämää GMP ohjeiston osan neljä liitettä 11
FDA	Food and Drug Administration, Yhdysvaltain elintarvike- ja lääkevirasto.
Fimea	Finnish Medicines Agency, Suomen lääkealan turvallisuus- ja kehittämiskeskus.
GAMP	Good Automated Manufacturing Practice
GCP	Good Clinical Practice
GDP	Good Distribution Practice
GLP	Good Laboratory Practice
GMP	Good Manufacturing Practice
GVP	Good Pharmacovigilance Practices
GxP	Tarkoittaa kollektiivisesti kaikkia hyviä tuotannollisia tapoja, kuten GMP, GCP, GLP, GDP, GVP (ISPE 2nd 2022, 12)
ISPE	The International Society for Pharmaceutical Engineering
IT Infra	Tietotekninen laitekokonaisuus, jolla mahdollistaa tietojärjestelmätoiminnallisuuksia (tietokoneet, tietoverkot, tietoturva ja tietovarastot)
MVP	Minimun viable product (pienin toimiva kokonaisuus)
OTS	Off-The-Self (valmisohjelmistoa kuvaava termi)
Part 11	Tarkoitetaan U.S.:n CFR:n Title 21, Part 11 kappaletta
PIC/S	Pharmaceutical Inspection Convention and Pharmaceutical Inspection Cooperation Scheme
QRM	Quality Risk Management (Laaturiskien hallinta)
REST	Representational state transfer
RMA	Risk Management Assessment (Riskienhallinta)
RUP	Rational Unified ProcessSDLC Software Development Life Cycle (Sovelluskehityksen elinkaari)
UAT	User Acceptance Testing (Käyttäjähyväksyntätestaus)
UML	Unified Modeling Language (Yhtenäinen mallinnus kieli)

1. JOHDANTO

Teollisuus on digitaalisessa murroksessa, jossa tekoälyä ja muita nousevia teknologioita pyritään valjastamaan osaksi liiketoiminnallisia prosesseja ja sitä kautta luomaan edellytyksiä kilpailuedulle. Digitalisaation tuomia suuria haasteita ovat jatkuvasti muuttuvat liiketoimintatarpeet ja datan käsittelyn räjähdysmäisen määrän lisääntyminen. Edellä mainittujen lisäksi, haastetta käyttöönotolle asettaa digitalisaation edellyttämä tarve integroida erilaisia tieto- ja ohjausjärjestelmiä jatkuvien tietovirtojen toteuttamiseksi. Hyvien tuotantotapojen toimintaympäristö (Good Manufacturing Practice), kuten lääketehtaat, tuo järjestelmäintegraatioiden toteutuksille omat lisähaasteensa, koska lääketehtaiden tietokoneistetut järjestelmät on validoitava. Validointi on dokumentoitu arvio käyttöönotettavan tietokoneistetun järjestelmän soveltuvuudesta aiottuun käyttöön, joka perustuu tieteelliseen näyttöön. Edellä kuvattu asetelma, kuten jatkuvat muutokset liiketoimintaprosesseissa, haastavat lääketeollisuuden perinteisiä validointiprosesseja ketterämmiksi.

Työn tutkimusasetelma on konstruktiivinen ja työllä oli tavoitteena vastata kahteen kysymykseen:

- Onko ohjelmistotestausautomaatio ja siihen liittyvän ketterän ohjelmistokehitysprosessimallin soveltuva (DevOps) lääketeollisessa toimintaympäristössä validointityön tukena?
- Onko mahdollista toteuttaa testausautomaatiota hyödyntävä automatisoitu työnkulku järjestelmäintegraation validointityön tueksi, sekä ennalta päätetyillä, että työn aikana valituilla työkaluohjelmistoilla?

Tavoitteisiin pääseminen edellytti perehtymistä lääketeollisuuden sääntelyn piirissä olevien tietokoneistettujen järjestelmien ja validointityöhön liittyviin keskeisiin hyvien tuotantotapojen (GMP) vaatimuksiin, sekä testaukseen ja testausautomaatioon liittyvien käsitteisiin ja prosessimalleihin, kuten DevOps-viitekehykseen sisältyvien jatkuvan integraation ja jatkuvan toimittamisen työkaluihin ja malleihin.

Validointityön ja DevOps prosessien yhdistämiseksi suoritettiin erilaisia sovellusasennuksia, konfigurointeja ja testejä, sekä kehitettiin ohjelmistoprototyyppejä esisuunnitteluvaiheessa käyttöön valituilla ohjelmistoilla säädellyn toimintaympäristön järjestelmäintegraatiosovellutuksessa. GMP-vaatimukset koskivat sekä teknisiä kyvykkyyksiä, että

hallinnollisia toimintoja. Työn tulokset esittävät testausautomaatiolla tuetun GMP-vaatimuksia tukevan DevOps elinkaarimallin, jota sovelletaan tilaajan järjestelmäintegraatiosovellukselle.

ASIASANAT:

järjestelmäintegraatio, säädelty liiketoiminta, validointi, ohjelmistotestaus, testausautomaatio, GMP, GxP, DevOps, CI/CD

2. SÄÄDELTY TOIMINTAYMPÄRISTÖ

Suomessa lääkevalvontaa suorittaa Lääkealan turvallisuus- ja kehittämiskeskus Fimea. Valvonta kattaa lääkkeen koko elinkaaren prekliinisestä vaiheesta lääkkeen vähittäisjakeluun asti. Lisäksi valvonta kattaa myyntiluvan haltijoiden lääketurvatoiminnan ja -markkinoinnin. (Fimea Valvonta n.d.) Fimea suuntautuu aktiivisesti kansainväliseen yhteistyöhön osana eurooppalaista viranomaisverkostoa (Fimea Tietoa Fimeasta n.d.), European Medicines Agency:ssä (EMA n.d.) jäsenyytensä välityksellä. Lääkkeiden tuotantoa ohjataan Fimean vahvistamilla viranomaismääräyksillä. Erityisesti lääkkeen valmistusta ohjataan lääkkeiden hyvillä tuotantotavoilla, englanniksi Good Manufacturing Practice, josta lyhenne GMP. GMP:llä varmistetaan lääkkeille kaikkien asetettujen vaatimusten täyttyminen valmistuksen osalta, jotka seuraavat hyvin pitkälti europan komission säätämiä direktiivejä ja tarkemmin komission julkaisemassa oppaassa ”Guide to Good Manufacturing Practice”. (Fimea 2022, 5–6; EudraLex Volume 4 n.d.)

Yhdysvalloissa lääkeviranomaisena toimii Food and Drug Administration, myöhemmin FDA, jonka liittovaltion säännöstö, aihe 21, sääntelee elintarvike- ja lääketeollisuutta (FDA 2022). Tämä ohjesääntö koskettaa myös Yhdysvaltojen ulkopuolelle sijoitettuja toimijoita, niiltä osin, jotka toimittavat elintarvike- ja lääketuotteita Yhdysvaltoihin (U.S Government 2007, 121 STAT. 853).

2.1 Lääkkeiden hyvät tuotantotavat (GMP)

Lääkkeiden hyvät tuotantotavat, GMP, määrittää lääkevalmistajille minimistandardin, joka valmistajan tulee täyttää tuotantoprosesseissaan. Kaikkien EU:n markkinoille tarkoitettujen lääkkeiden valmistajien on noudatettava GMP:tä, riippumatta siitä, missä lääkkeet valmistetaan. GMP vaatii, että lääkkeet ovat tasaisen korkealaatuisia, ovat tarkoitukseen sopivia ja täyttävät myyntiluvan tai kliinisen tutkimuksen luvan vaatimuksia. (EMA 2022.)

Tämän työn kannalta olennaisia GMP vaatimuksia ovat vaatimukset liittyen sähköisiin tietojenkäsittelymenetelmiin, niistä FDA:n pykälä Title 21 CFR part 11 – Electronic Records; Electronic Signatures, myöhemmin Part 11, koskee erityisesti vaatimuksia mm. sähköisten tallenteiden osalta, kuten pykälässä §11.1(b) säädetään (21 CFR 1997). Vastaavasti Euroopan komissio on säätänyt direktiivin tietokoneistettuihin järjestelmiin liittyen, EudraLex, The Rules Governing Medicinal Products in the European Union, Volume 4, Good Manufacturing Practice, Medicinal Products for Human and Veterinary

4 Good Manufacturing Practice Annex 11 2011. Jossa säädetään hyvin samaan tapaan kuin FDA:n Title 21 CFR part 11 säädöksessä, mutta se ei rajoitu vain elektronisiin tallenteisiin (Pavlović 2020). Annex 11 sisältää vaatimuksia koko järjestelmälle ja rakentuu riskienhallinnan näkökulmasta, kun taas FDA nojaa lähinnä tietoturvanäkökulmiin (EduQuest 2011, Table 1). Tässä työssä aihetta käsitellään lähinnä EU Annex 11 ohjeistuksen mukaisesti.

GMP oppaat ja viranomaisdokumentit usein viittaavat yleisesti lyhenteeseen GxP, tietokoneistettujen järjestelmien yhteydessä, jolla tarkoitetaan kaikkia farmaseuttisia ”hyviä tapoja”, kuten Good Clinical Practice (GCP), Good Laboratory Practice (GLP), Good Distribution Practice (GDP) ja Good Pharmacovigilance Practices (GVP) (ISPE 2nd 2022, 12). GMP termiä käytetään, kun kyseessä on lääketuotanto- tai lääkeliiketoimintaprosessi, ja GxP termiä, kun kyseessä on tietokoneistettu järjestelmä, joka tukee GMP prosessia jollakin farmaseuttisella alueella. Omiin kokemuksiini perustuen vakiintuneessa puhekielessä GxP-järjestelmällä usein tarkoitetaan yleisesti GMP toimialueella olevia tietojärjestelmiä.

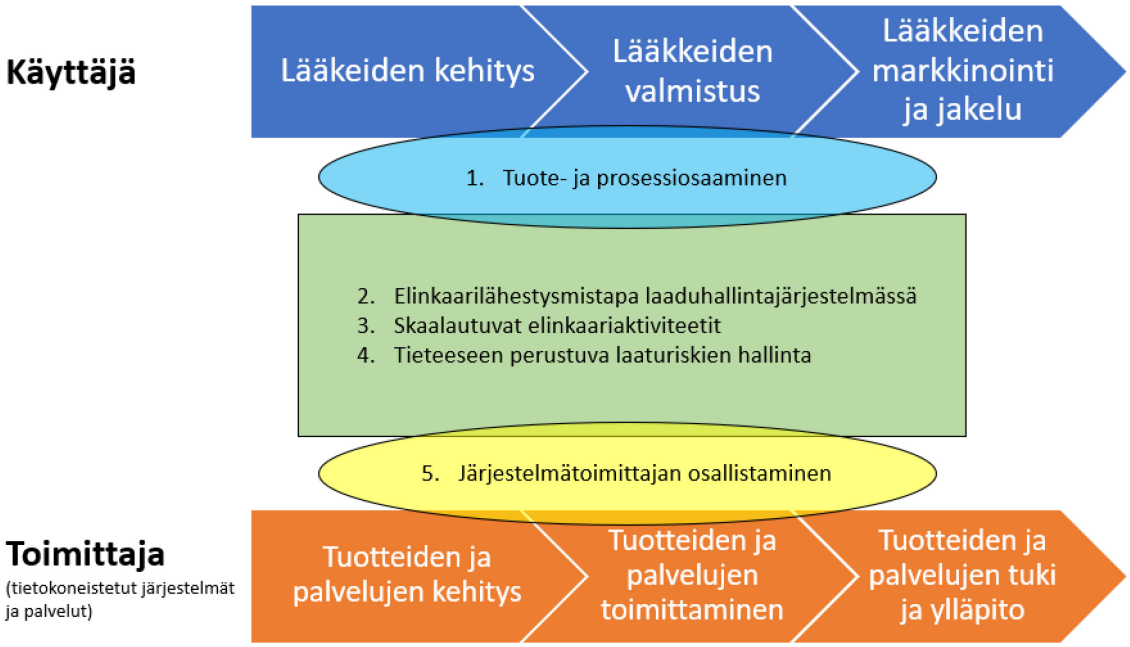
2.2 ISPE – The International Society for Pharmaceutical Engineering

ISPE on voittoa tavoittelematon yhdistys, joka palvelee jäseniään johtamalla tieteellistä, teknistä ja lainsäädännöllistä kehitystä farmaseuttisen koko elinkaaren ajan. ISPE on perustettu vuonna 1980 pienen pohjoisamerikkalaisen insinöörityöryhmän yhteistyön seurauksena, joka laajeni insinööritaitojen kehittämisen lisäksi laajasti eri lääkealan ammattilaisten edustajaksi. ISPE mahdollistaa mm. neutraalin ympäristön, jossa yksittäiset jäsenet ja sääntelyviranomaisiin kuuluvat asiantuntijat voivat käydä avointa vuoropuhelua, joista on lopulta hyötyä potilaille ympäri maailman. (About ISPE n.d.)

2.3 GAMP – Good Automated Manufacturing Practice

GAMP tarkoittaa tässä yhteydessä hyviä automaattisten tuotantolaitteiden ja -järjestelmien tuotantotapoja, joka on järjestelmä laadukkaiden tuotantolaitteiden tuottamiseksi käyttämällä prospektiivisen validoinnin käsitettä elinkaarimallin mukaisesti. Suunniteltu erityisesti laitetoimittajille ja laitteiden käyttäjille lääketeollisuudessa. (Glossary ISPE. n.d.) ISPE julkaisee erilaisia lääketeollisuuden ammattilaisten tuottamia hyvien käytäntöjen oppaita, jotka tarjoavat käytännönläheistä tietoa parhaista lääketeollisuuden käytännöistä ja sääntelyn odotuksista ja auttavat kaventamaan erilaisia tulkintoja sääntelystandardeista (Glossary ISPE n.d.). GAMP 5 ohjeistuksen tavoitteena on ohjeistaa, miten saadaan lääketeollisuuteen yhteensopivia tietokoneistettuja järjestelmiä.

(Guidance Documents ISPE n.d.) GAMP 5 ei kuitenkaan anna eväitä varsinaiseen järjestelmien suunnittelutyöhön, ts. ei ole menetelmä tai standardi (ISPE 2008, 11). GAMP 5 on konsepti millä käyttäjäorganisaatio ja järjestelmän toimittaja saadaan toimimaan yhteistyössä, jotta vaatimukset saadaan täytettyä. Konsepti käsitetasolla kuvattuna alla (Kuva 1).



Kuva 1. *ISPE GAMP 5:n viisi avainkonseptia (mukaillen, ISPE 2008, 19)*

GAMP 5 keskittyy riskikeskeisesti ohjaamaan toteutusprojektia, osana yrityksen (käyttäjän/tilaajan) laatujärjestelmää, kannustamalla ymmärtämään liiketoimintaprosessia, määrittelemään käyttäjävaatimukset ja toteuttamalla ne systemaattisesti elinkaariajattelun avulla vahvassa toimittajayhteystyössä, niin että toteutus vastaa dokumentoidusti käyttäjävaatimuksia, sekä sääntelyn vaatimuksia. GAMP 5 ei siis kerro miten jokin sääntelyvaatimus tulisi toteuttaa ja minkälaisia menetelmiä ohjelmistokehityksessä tulisi käyttää. (ISPE 2008, 14–15.)

Konseptin keskeisinä riskinäkökulma-ajureina toimivat potilasturvallisuus, tuotteiden laatu (lääkkeet ja lääkinnälliset laitteet) ja tiedon eheys (ISPE 2008, 14). Lääkeliiketoimintaa harjoittava yritys määrittää laatujärjestelmässään mitä ohjeita ja toimintatapoja noudatetaan tietokoneistettujen järjestelmien validoinnissa. Tämän työn kannalta GAMP 5 on olennainen, koska tilaaja noudattaa GAMP 5 ohjeistuksen käytäntöjä osana laatujärjestelmäänsä. GAMP 5 käytännönläheisen lähestymistavan ja käyttökelpoisuuden myötä siitä on syntynyt de-facto standardi lääkealalle, koska maailmanlaajuisesti hyvin monet toimijat hyödyntävät ohjeistusta toiminnassaan. (Soinio Hanna 2022).

Ensimmäinen GAMP 5 opas julkaistiin vuonna 2008 (ISPE 2008). Samoihin aikoihin, kun tätä työtä on käynnistelty, ISPE on julkaissut uuden version GAMP 5 ohjeistuksesta (ISPE 2nd 2022). Ohjeistuksessa on julkaistu kokonaan uusia liitteitä seuraavilla osa-alueilla:

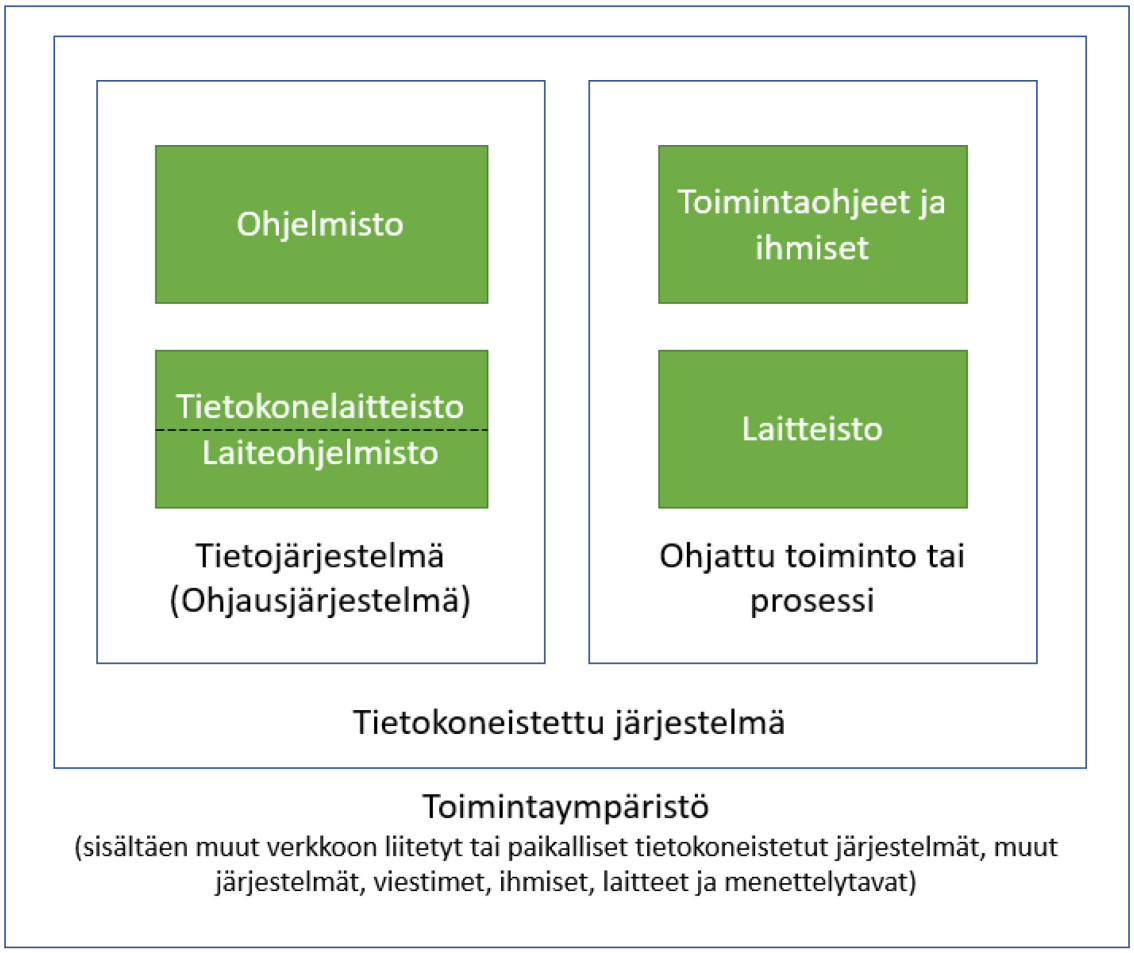
- Agile
- Software Tools
- Distributed Ledger Systems (Blockchain)
- Artificial Intelligence and Machine Learning (AI/ML)
- IT Infrastructure
- Critical Thinking (ISPE 2nd 2022, 11).

Uudet osa-alueet kattavat nyt edellisen version puutteet mm. nousevien teknologioiden osalta, kuten älykkäät järjestelmät, koneoppiminen ja ketterät menetelmät.

GAMP 5 ohjeistuksen pyrkimys siihen, että tilaaja ja toimittaja toteuttavat ratkaisut dokumentoidusti tiiviissä yhteistyössä on ajatusmaailmaltaan on hyvin samankaltainen monien ohjelmistokehityselinkaarimallien (SDLC) kanssa, kuten Agilessa (Agile Model Tutorialspoint n.d.) nojataan vahvaan yhteistyöhön, Spraalimallissa (Spiral Model Tutorialspoint n.d.) toiminta on hyvin riskikeskeistä, vesiputousmallissa (Waterfall Model Tutorialspoint n.d.) nojataan hyvään dokumentoituun suunnitteluun ennakolta.

2.4 Tietokoneistetut järjestelmät lääketeollisuudessa

Kansalliskirjaston Suomalainen asiasanasto- ja ontologiapalvelu määrittää, että tietojärjestelmä on kokonaisuus, joka koostuu tietoja käsittelevistä ihmisistä, tietoihin liittyvistä laitteista, ohjelmistoista ja tietojenkäsittelysäännöistä, joka mahdollistaa toimintoja ja toimintaa, niitä tehostamalla ja helpottamalla (Kansalliskirjasto 2018). Viranomaisten GMP ohjeistukset käyttävät pääsääntöisesti termiä ”Computerized System”, joka käännettynä suomeksi on tietokoneistettu järjestelmä. EU Annex 11 määrittää tietokoneistetun järjestelmän kokoelmaksi laitteita ja ohjelmistoja, jotka yhdessä täyttävät joitakin toiminallisuusvaatimuksia, mutta sellaisessa asiayhteydessä, että sitä käytetään osana GMP-säänneltyä toimintaa (EudraLex Volume 4 Good Manufacturing Practice Annex 11 2011, 2). Vastaavasti Part 11 käyttää termiä ”Computer Systems” viranomaistarkastuksiin liittyvässä määräyksessä (21 CFR 1997, 11.1(e)). PIC/S:n määritelmän mukaisen tietokoneistetun järjestelmän kaavio riippuvuussuhteineen seuraavassa kuvassa (Kuva 2).



Kuva 2. *Eri komponenttien riippuvuudet tietokoneistetun järjestelmän toimintaympäristössä (mukaillen, PIC/S Secretariat, 8).*

ISPE:n GAMP 5 ohjeistuksen mukaan tietokoneistettu järjestelmä koostuu laite-, sovellus- ja verkkoliikennekomponenteista yhdessä kontrolloitujen toiminnallisuuksien ja dokumentaation kanssa. Termi sisältää laajan alueen erilaisia järjestelmiä, kuten automaattisen valmistuksen laitteet ja ohjausjärjestelmät ts. automaatiojärjestelmät (ISPE 2008, 21–22). Voidaan päätellä, että Kansalliskirjaston tietojärjestelmä määritelmällä tarkoitetaan samankaltaista kokonaisuutta kuin GMP:ssä tarkoitetaan tietokoneistetulla järjestelmällä, mutta GMP tuntee tietojärjestelmän terminä vain osana tietokoneistettua järjestelmää. On kuitenkin vähintäänkin hämmentävää, että englanninkielisissä materiaaleissa saatetaan käyttää eri termejä mutta tarkoitetaan samaa asiaa. Tilanne voi olla sama myös Suomen kielessä, tietojärjestelmä ja tietokoneistettu järjestelmä termien eroja ei GxP-mielessä välttämättä edellä kuvatusta syystä osata erottaa.

2.5 Tietokoneistetulta järjestelmältä vaadittavat ominaisuudet

Lyhyesti, sovellukset tulee validoida ja IT infrastruktuuri tulee kvalifioida (EudraLex Volume 4 Good Manufacturing Practice Annex 11 2011, 2). Käytännössä tämä tarkoittaa

sitä, että todennetaan erilaisin menetelmin, että käytetyt järjestelmät täyttävät vaatimukset ja toimivat suunnitellulla tavalla. Keskeiset GxP-järjestelmän vaatimukset EU Annex 11 (EudraLex Volume 4 Good Manufacturing Practice Annex 11 2011, 3–5.) mukaan ovat lueteltuna taulukossa alla (Taulukko 1).

Tallenteet (Data)	Tietokoneistetun järjestelmän vaihtaessa tallenteita elektronisesti toisen järjestelmän kanssa tulee tiedonsiirtoon liittyä sisäänrakennettuja tarkastuksia tallenteen oikeellisuuden varmistamiseksi ja riskien minimoimiseksi.
Oikeellisuuden tarkastus (Accuracy Checks)	Manuaalisesti syötettävän kriittisen tiedon yhteydessä pitää olla lisätarkastuksia tiedon oikeellisuuteen liittyen, joko toisen käyttäjät toimesta tai sähköisin menetelmin. Mahdollisen väärän tiedon aiheuttamat riskit tulee kattaa riskien hallinnalla.
Tietovarasto (Data Storage)	Tallenteet tulee turvata sekä fyysisiltä, että elektronisilta uhkilta. Tallennetun tiedon saatavuus, luettavuus ja paikkansapitävyys tulee tarkastaa. Saatavuus tulee varmistaa koko tallenteen elinkaaren ajan. Tietovarastot tulee varmistaa säännöllisesti. Varmuuskopioiden oikeellisuus ja kyky palauttaa tietoja tulee varmistaa validoinnin yhteydessä ja tarkastaa määräajoin.
Tulosteet (Printouts)	Tietojen tulee olla mahdollista tulostaa ulos järjestelmästä, ymmärrettävässä muodossa.
Aukoton kirjausketju (Audit Trail)	Riskienhallintaan perustuen järjestelmän tulee pystyä jäljittämään GMP-relevantteihin tietoihin liittyvät muutokset ja poistamiset, joihin on dokumentoitu syy muutokselle. Kirjausketju pitää olla saatavilla avoimesti ja ymmärrettävässä muodossa, sekä säännöllisesti tarkastettuna.
Muutoksien ja konfiguraation hallinta (Change and Configuration Management)	Kaikki järjestelemään kohdistuvat muutokset pitää toteuttaa kontrolloidusti määritetyn prosessin mukaisesti.
Määräaikaistarkastukset (Periodic Evaluation)	Järjestelmät tulee säännöllisesti tarkastaa ja varmistua siitä, että ovat validoidussa tilassa ja täyttävät GMP vaatimukset.
Tietoturva (Security)	Järjestelmään tulee olla pääsy vain niihin oikeutetuilla henkilöillä ja järjestelmä tulee turvata fyysisin ja/tai loogisin menettelyin. Oikeuksia voidaan hallita avaimilla, kulkukorteilla, henkilökohtaisilla tunnuksilla ja salasanoilla, sekä biometrisillä tunnisteilla.
Häiriöhallinta (Incident Management)	Kaikki häiriöt, ei pelkästään järjestelmä- tai tietohäiriöt, tulee raportoida ja arvioida. Kriittisten häiriöiden juurisyyt tulee selvittää ja käsitellä korjaavina ja ehkäisevinä toimenpiteinä.

Elektroninen Allekirjoitus (Electronic Signatures)	Tallenteita voidaan allekirjoittaa elektronisella allekirjoituksella. Elektronisen allekirjoituksen tulee vastata käsin kirjoitettua allekirjoitusta ja linkittyä pysyvästi allekirjoitettuun tallenteeseen ajankohta mukaan lukien.
Erän vapautus (Batch release)	Jos järjestelmää käytetään erävapautukseen (tuotteiden vapautus markkinoille). Järjestelmän tulee sallia vain tähän pätevöitetyn henkilön suorittaa toiminnallisuus ja järjestelmän tulee tarkasti tallentaa henkilön tiedot, joka vapautuksen tekee. Erävapautuksessa tulee käyttää sähköistä allekirjoitusta.
Liiketoiminnan jatkuvuus (Business Continuity)	Liiketoiminnan jatkuvuus tulee varmistaa vaihtoehtoisin järjestelyin (manuaalisesti tai toisella järjestelmällä) tilanteessa, jossa järjestelmä on vikaantunut. Aika missä vaiheessa vaihtoehtoinen järjestelmä otetaan käyttöön, tulee perustua riskiarviointiin. Järjestely tulee olla dokumentoituna ja testattuna.
Arkistointi (Archiving)	Tiedot voidaan arkistoida ja se tulee tarkastaa saatavuuden, luettavuuden ja eheyden osalta. Jos järjestelmään kohdistuu merkittäviä muutoksia, tulee arkistoiden toimivuus varmistaa ja tarkastaa.

Taulukko 1. *Keskeiset EU-direktiivin mukaiset GxP vaatimukset tietokoneistetuille järjestelmille (mukaillen, EudraLex Volume 4 Good Manufacturing Practice Annex 11 2011).*

Tämä lisäksi GxP-järjestelmiin liittyy yleisiä toimintaa liittyviä vaatimuksia, kuten manuaalisen toiminnan korvautuessa tietokoneistetulla toiminnalla, kokonaisriski ei saa nousta. Riskien hallinta tulee olla keskeinen prosessinhallinnan väline koko järjestelmän elinkaaren aikana, joka huomioi potilasturvallisuuden, tiedon eheyden ja tuotteiden laadun. GxP-järjestelmille pitää olla nimettynä prosessin omistaja ja järjestelmän omistaja, sekä kaikille järjestelmän käyttäjillä pitää olla ajantasainen pätevyys, määritetty rooli ja roolin mukaiset käyttöoikeudet. Kaikki järjestelmään liittyvät kolmannet osapuolet pitää sitoa sopimusteitse noudattamaan samoja laatuvaatimuksia. (EudraLex Volume 4 Good Manufacturing Practice Annex 11, 2.)

3. KETTERÄ OHJELMISTOKEHITYSMENETELMÄ

Kappaleessa perehdytään ketterään ohjelmistokehitykseen liittyviin käsitteisiin, sekä DevOps-viitekehykseen, joka pyrkii toteuttamaan ketterän ohjelmistokehityksen elinkaarta. Tämän lisäksi tutkitaan näiden menetelmien tuomien prosessien yhteensopivuutta lääketeollisuuden tarpeisiin, kuten automaattinen testaaminen.

3.1 Ketterä ohjelmistokehitys

Ketterä (Agile) ohjelmistokehitys, joka perustuu 17 itsenäisen ohjelmistoalan ammattilaisen kirjoittamaan manifestiin, joka koostuu neljästä ydinarvosta ja 12 toimintaperiaatteesta. Ydinarvot ovat,

- enemmän yksilöitä ja yhteistyötä kuin menetelmiä ja työkaluja
- enemmän toimivaa ohjelmistoa kuin kattavaa dokumentaatiota
- enemmän asiakasyhteistyötä kuin sopimusneuvotteluja
- enemmän vastaamista muutokseen kuin pitäytymistä suunnitelmassa.

Manifesti lisäksi mainitsee, että vaikka näiden neljän ydinarvon jälkimmäisillä asioilla on myös arvoa, niin se arvostaa ensin mainittuja enemmän. Toimintaperiaatteissa korostetaan asiakastyytyväisyyttä, muutoskyvykkyyttä, toimivaa ohjelmistoa, yhteistyötä ja toiminnan omaa arviointia. (Agile Alliance n.d.)

Käytännössä ketterä ohjelmistokehitys on yhdistelmä iteratiivisen ja inkrementaalisen ohjelmistokehityksen elinkaarimalleista. Ketterää ohjelmistokehitystä voidaan toteuttaa kokoelmalla erilaisia metodeja, kuten esimerkiksi Rational Unified Process (RUP), Scrum ja Extreme Programming. (Agile Model Tutorialspoint. n.d.) Ketterä ohjelmistokehitys korostaa käyttäjäyhteistyötä (Lwakatare 2017, 19) ja sitä pidetään hyvänä lähtökohtana DevOps käytäntöihin siirtymisessä (Lwakatare 2017, 33).

3.2 DevOps

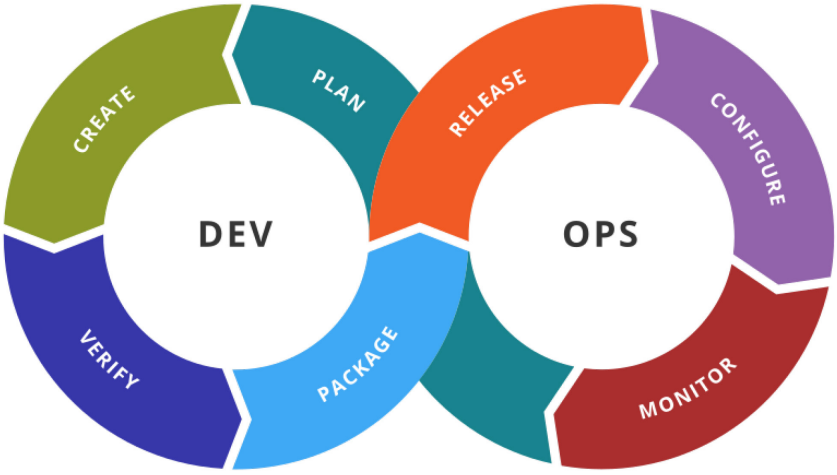
DevOps vie ketterää ohjelmistokehitysprosessia pidemmälle yhdistämällä IT-operaatiot osaksi elinkaarta. Sitä voisi kuvailla toimintamalliksi tai konseptiksi, jonka päällimmäinen pyrkimys on nopeuttaa ohjelmistotuotantoa hyödyntämällä automaation, mittaamisen, monitoroinnin ja yhteystyön eri ulottuvuuksia. DevOps termistä ei kuitenkaan ole selkeää määritelmää, eikä se ole standardi, jota seurata, mutta ajatusmalliltaan ”Dev”

tarkoittaa sovelluskehitystä ja ”Ops” sovellukseen liittyviä operatiivisia tehtäviä. DevOps termi korostaa kulttuurillisia muutoksia, kuten luottamuksen, yhteenkuuluvuuden ja vastuiden rakentamisen kehittäjien ja järjestelmävastaavien välillä, mutta DevOps ei määritä tarkasti minkälainen organisaation tulisi olla. DevOps toimintatapoihin siirtyminen aiheuttaa muutoksia ohjelmistojen toimitusketjuihin, palveluihin, työtehtäviin, IT-työkaluihin ja parhaisiin käytäntöihin. (Lwakatare 2017, 25–36.)

Tyyppillisiin DevOps työtapoihin kuuluvat jatkuvan integraation (CI) ja jatkuvan toimituksen (CD) menetelmät, eli CI/CD-putket. Prosessien lisäksi versionhallintatyökalut lähdekoodeille ja tuotoksille, sekä ohjelmistoratkaisun mukaan muita hallintatyökaluja, kuten säiliöt, konfiguraation hallinta tai pilviympäristöt. (Lwakatare 2017, 36–37; Courtemanche, Mell & Gills n.d.) DevOps laajentaa ketterää ohjelmistokehitystä siinä, että DevOps:n avulla pyritään automatisoimaan kaikki tarvittavat toimenpiteet ohjelmiston toimittamiseksi tuotantoon asti (Packer 2022). Edellä kuvatut toimitusputket ovat ohjelmistokehityksen ja operaatioiden välinen yhdistävä tekijä, joiden tehtävänä on varmistaa, että tehdyt muutokset päätyvät tuotantoon testattuina ja toimivina (Lwakatare 2017, 36–38).

Operaatioiden osalta DevOps mallilla pyritään löytämään keinoja tunnistaa tuotannossa esiintyvien ongelmien juurisyitä kehityksessä tehtyihin päätöksiin, esimerkiksi etsimällä korrelaatioita suoritusdatan ja tehtyjen muutoksien välillä. Lisäksi operaatioihin voisi kuulua erilaisten monitorointityökalujen käyttö ohjelmiston häiriöiden eskaloimiseksi tai palvelujen uudelleenkäynnistämiseksi. Työkaluja voisi olla myös käyttäjätutkimuksen automatisoimiseksi tai sovellusympäristön suorituskyvyn mittaamiseksi erilaisissa kuormitustilanteissa. (Lwakatare 2017, 39–40.)

DevOps-prosessia voisi kuvata ikään kuin päättymättömänä silmukkana (Kuva 3), joka sisältää seuraavat vaiheet: suunnittelu, koodi ja kokoaminen, testaus, julkaisu, käyttöönotto, käyttö ja valvonta, sekä palautteen käsittelyn kautta uusi suunnitelma seuraavalle iteraatiokierrokselle. (Courtemanche, Mell & Gills n.d.)

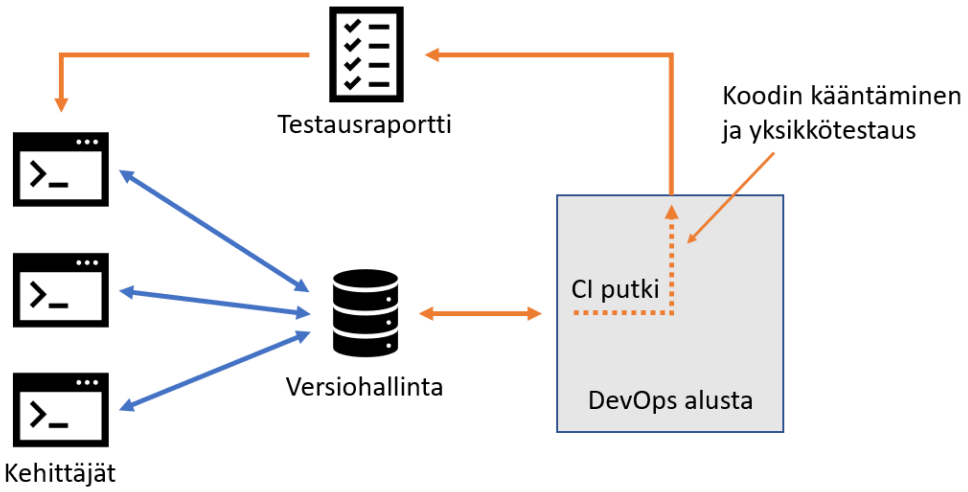


Kuva 3. *DevOps silmukka (https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg)*

DevOps hyötyjä ovat toteutuksien nopeus ja laatu, erityisesti muutostilanteissa, sekä parantunut kommunikointi eri sidosryhmien välillä. Haasteena vastaavasti on päästä eroon vanhoista tottumuksista. Siiloutuneet organisaatiot ja tiettyihin tehtäviin erikoistuneet tiimit voivat kamppailla DevOps-käytäntöjen omaksumisessa. Toisaalta voi syntyä myös harhaluulo, että pelkästään uusien työkalujen käyttöönotto riittää, ilman panostusta kouluttamiseen. Jotta DevOps onnistuu, on jokaisen siinä työskentelevän ymmärrettävä koko arvoketju, ideasta kehityksen kautta loppukäyttäjäkokemukseen. Ennen kaikkea DevOps työkulttuurellinen muutos, joka edellyttää sitoutumista kaikilta, jotka siihen osallistuvat. (Lwakatare 2017, 41–44; Pihlajamäki n.d.) Edellä mainittujen haasteluuksien lisäksi DevOps:n toimitusputkien käyttöönotto voi olla hankalaa vanhentuneita teknologioita hyödyntäville toteutuksille (legacy software), mutta myös sellaisille toteutuksille, jotka sijoittuvat säädeltyyn toimintaan, lähinnä, soveltuvien työkalujen puutteen takia (Lwakatare 2017, 45–46).

3.3 Testaus- ja toimitusputket

Testaus- ja toimitusputket, eli CI ja CD putket ovat DevOps:n keskeisiä työkaluja, joilla voidaan automatisoida ohjelmistotuotteen kehitystyötä. CI eli jatkuva integrointi (Continuous Integration) on jatkuva käytäntö, jossa tiimin kaikkien kehittäjien koodimuutokset yhdistetään päälähdekoodiin mahdollisimman usein ja sen käyttäytyminen testataan automaattisesti (Virmani 2015, 79 & Prince 2016). CI putki vie ketterää ohjelmistokehitystä askelta pidemmälle automatisoimalla koodien integroinnin (koonnin) ja testauksen (Packer 2022). Toimivan CI:n (Kuva 4) edellytys on versionhallintaohjelmisto ja CI prosesseja tukeva palvelu, joka osaa kääntää, ajaa testitapauksia ja luoda raportin tuloksista (Courtemanche, Mell & Gills n.d.).

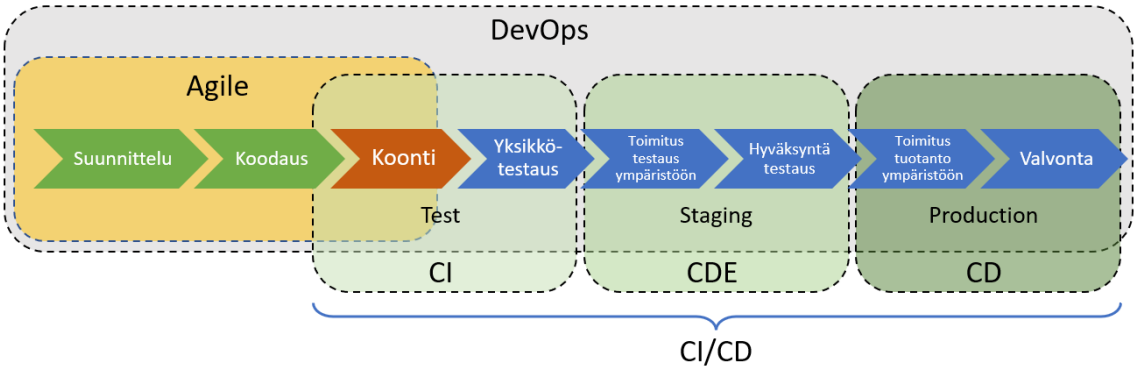


Kuva 4. *Mahdollinen CI-putken työnkulku.*

CD eli jatkuva toimittaminen (Continuous Delivery) on käytäntö, jossa CI prosessin tuottamat tuotokset toimitetaan automaattisesti testattuna käyttöympäristöön. CD:llä voidaan tarkoittaa myös jatkuvaa käyttöönottoa (Continuous Deployment) (Virmani 2015, 79). Näiden erottelemiseksi, joissakin yhteyksissä käytetään termiä CDE jatkuvalle toimittamiselle. Termeillä CDE ja CD pyritään erottelemaan vaiheet tuotoksien toimittamisesta erilliseen testausympäristöön (Staging) ja tämän jälkeen varsinaiseen tuotantoympäristöön (Production) (Packer 2022 & Prince 2016). Vastaavasti IEEE access artikkelissa (Mojtaba & Muhammand & Liming 2017) CDE:lla tarkoitetaan toimitusputkea, jossa varsinaisesta testausympäristöstä toteutus toimitetaan tuotantoympäristöön manuaalisesti ja CD:llä vastaavasti täysin automaattista toimitusputkea. ts. CDE ja CD ovat rinnakkaiset prosessit eri käytännöillä. Samaisessa artikkelissa todetaan, että CDE vaiheella tarkoitetaan kehitysvaihetta, jossa toteutusta pidetään tuotantovalmiina, mutta sitä ei ole vielä viety varsinaiseen tuotantojärjestelmään. Tässä työssä CDE:llä nimenomaan tarkoitetaan jälkimmäistä, eli tuotantovalmista, joka sijaitsee testausjärjestelmässä.

3.4 Ketterä kehitys (Agile), DevOps ja CI/CD yhdessä

Ketterän ohjelmistokehityksen (Agile) tavoitteena on huolehtia varsinaisen kehitystyön prosesseista, kuten määritellä sovellukseen kehitettävät ominaisuudet, koordinoida työtä ja tehtäviä, sekä kerätä palautetta toteutuksista. Vastaavasti CI/CD:n rooli on automatisoida koodin kirjoittamisen jälkeen tapahtuvia toimenpiteitä mahdollisimman paljon, jolla saadaan uudet ominaisuudet tuotantoon hallitusti ja testatusti toimivana. Tämä on kokonaisuus, joka muodostaa DevOps käsitteen.



Kuva 5. *Agile ja CI/CD DevOps kehityselinkaaressa (mukaillen Packer 2022 & Prince 2016)*

Edeltävässä kuvassa (Kuva 5) Agile ja CI/CD sijoitettuna sovelluksen kehityselinkaaren ei vaiheille ja alustoille. Kehitystyötä ei ole pakko toteuttaa Agilea ja CI/CD:tä soveltamalla, mutta automatisoimalla elinkaaren vaiheita suoriudutaan monista tehtävistä paljon nopeammin ja aina samalla suorituskyvyllä, esimerkiksi jatkuvasti toistettavien testien osalta (Virmani 2015, 79).

3.5 Ketterä kehitys lääketeollisuudessa

ISPE:n mukaan (ISPE 2nd, 245) ajurina ketterien menetelmien tunnustamiseen on FDA:n TMAP (Technology Modernization Action Plan) suunnitelma, jossa tähdätään siihen, että FDA:n tarkastustoiminta olisi valmis entistä datarikkaampiin ympäristöihin biolääketieteen alalla (TMAP 2019, 1). He mainitsevat mm. ”DevOps”:in ongelmakeskeiseksi sovelluskehitys menetelmäksi (TMAP 2019, 2). Voidaan siis olettaa, että viranomaiset ovat myös havainneet ketterät sovelluskehitysmenetelmät olevan osa sääntelyn piirissä olevien yritysten toimintaa. GAMP 5 liitteessä D8 (ISPE 2nd 2022, 245–251) käydään systemaattisesti läpi eri GMP-näkökulmia Agile SDLC prosessien kannalta.

Koska ketterässä kehityksessä on pohjimmiltaan kyse kyvystä luoda uusia ominaisuuksia (muutoksia) ja vastata niihin kontrolloidusti, niin se vaikuttaa GxP-hallinnan ja ylläpitämisen näkökulmasta haasteelta. Toisaalta, jotta ketterä kehitys toimii oikein, on siinä käytettävien prosessien oltava selkeitä ja hyvin hallittuja, sekä työkalut, joilla sen voi osoittaa. Edellä mainittujen lisäksi tiimillä on oltava vastuu työkalujen oikeasta käytöstä ja lopulta koko toteutuksesta. (ISPE 2nd 2022, 251.)

GAMP 5 oppaassa todetaan, että ketterä kehitys oikein käytettynä täyttää GxP:n tarkoittamat hallinnan ja valvonnan tarpeet, koska ketteryyden keskeinen tavoite on toimittaa vain toimivia ja testattuja ohjelmistoja (ISPE 2nd 2022, 251). Tätä ajatusta tukee se, että viranomaismääräyksien osalta (EudraLex Volume 4 Good Manufacturing Prac-

tice Annex 11 2011) säädetään vain, että validointia varten tulisi olla prosessi, joka varmistaa laadun ja suorituskyvyn muodollisen arvioinnin ja raportoinnin toimenpiteet järjestelmän kaikille elinkaaren vaiheille.

4. TESTAUSAUTOMAATIO OHJELMISTOKEHITYKSEN TUKENA

Testausautomaatiolla tarkoitetaan järjestelyä, jossa ohjelman tai sovelluksen testaamista varten rakennetaan automaattisia työvälineitä ennalta suunniteltujen testien tekemiseksi. Testausautomaation tavoitteena on rakentaa järjestely, jossa tietty joukko toistuvasti tehtäviä testitapauksia voidaan suorittaa nopeaa tarkastusta varten. (Kasurinen 2013, 60.) Testausstrategiaa ohjaa yrityksen laatujärjestelmä, joka on keskeinen ja pakollinen järjestelmä sääntelyn piirissä olevissa yrityksissä (EudraLex Volume 4 Good Manufacturing Practice Chapter 1 2013, 2). Tässä kappaleessa perehdytään erilaisiin testauskäsitteisiin yleisesti.

Testauskäsitteet kategorisesti jakaantuvat staattisiin ja dynaamisiin testeihin. Staattisella testaamisella tarkoitetaan järjestelmän tutkimista ja arvioimista ilman varsinaista järjestelmän käyttöä, kuten katselmoimalla koodia tai arvioimalla rakennettavan järjestelmän arkkitehtuuria. Dynaaminen testaaminen vastaavasti on testaamisen muoto, jossa varsinaista toteutettavan järjestelmän tai sen osien reaktioita testataan erilaisilla syötteillä. Erilaiset testausmenetelmät sijoittuvat ohjelmiston elinkaarelle eri kohtiin riippuen missä vaiheessa ohjelmiston kehitystä sijaitaan. Testauksen kannalta keskeiset vaiheet ovat, esituotanto, kehitys ja julkaisuvaihe. (Kasurinen 2013, 48–63.)

Esituotantovaiheessa testaaminen keskittyy käyttöliittymäsuunnitteluun ja varsinaiseen testauksen suunnitteluun, jotka perustuvat vaatimusmäärittelyyn. Kehitysvaiheessa testaaminen keskittyy toiminnalliseen testaamiseen ja ennen julkaisua testaaminen voi olla hyvin moninaista, mutta tyypillisesti se sisältää erilaisia kuormitus- ja suorituskykyyn liittyviä testejä, sekä käytettävyyteen liittyviä testejä. (Kasurinen 2013, 48–63.) Testausautomaation käyttö testaukseen tukena voi vähentää merkittävästi manuaalisesti testattavan regressiotestauksen määrää, mutta sen käyttäminen ei kuitenkaan täysin poista tarvetta manuaaliselle testaamiselle (ISPE 2nd 2022, 61).

4.1 Mustalaatikkotestaus

Kehitysvaiheessa mustalaatikkotestauksessa ohjelmalle tai sen osalle annetaan syötteitä ja tarkastetaan mitä ohjelma tekee tai miten se reagoi. Mustalaatikkotestauksessa ei nähdä varsinaisen ohjelman sisällä tapahtuvia asioita. (Kasurinen 2013, 51.)

4.2 Lasilaatikkotestaus

Lasilaatikkotestaus on täydellisempi versio mustalaatikkotestauksesta. Tässä menetelmässä nähdään myös mitä varsinaisen testattavan ohjelman sisällä tapahtuu. (Kasurinen 2013, 52.)

4.3 Harmaalaatikkotestaus

Harmaa- laatikkotestaus on lasilaatikkotestauksen ja musta- laatikkotestauksen yhdistelmä. Tilanne tulee esiin niissä tapauksissa, kun sovellus, jonka sisälle testauksen aikana nähdään, käyttää jotain ulkoista palvelua. Tällöin ulkoinen palvelu käyttäytyy musta laatikkotestauksen kaltaisesti, koska sen sisälle ei varsinaisesti nähdä. (Kasurinen 2013, 52–53.)

4.4 Regressiotestaus

Regressiotestaaminen ei varsinaisesti ole oma testauksen muotonsa, vaan sillä tarkoitetaan käytännössä uudelleentestaamista. Yleisesti regressiotestaamisella tarkoitetaan tilannetta, jossa järjestelmään tehdään muutoksia ja halutaan varmistaa, että järjestelmä toimii muilta osin edelleen halutulla tavalla. (Kasurinen 2013, 52–53.)

4.5 Käytettävyystestaus

Käytettävyystestaus kirjaimellisesti tutkii, miten hyvin ohjelmiston käyttöliittymän toimii käyttäjän kanssa, sekä toiminnalliselle tasolle, että intuitiotasolla. Tätä varten on kehitetty erilaisia menetelmiä, kuten videoseurantaa unohtamatta ihan perinteistä kyselytutkimista. (Kasurinen 2013, 55.)

4.6 Kuormitustestaus

Kuormitustestauksella pyritään simuloimaan ohjelmistolle suunnitellun käyttäjämäärän tai käyttötapauksen tuottama kuormitus ja tutkimaan aiheutuuko tästä ohjelmistolle mainittavia ongelmia, jotka voivat olla erilaisia pullonkauloja, kuten tietoliikenteeseen liittyvät vasteajat. Tämän lisäksi kuormitustestauksella saadaan selville ohjelmiston kapasiteetti ja miten se selviää normaaleista käyttöolosuhteista. (Kasurinen 2013, 55–56.)

4.7 Suorituskykytestaus

Suorituskykytestaus on vastaa kuormitustestausta, mutta se suoritetaan vaatimusmäärittelyä vasten (Kasurinen 2013, 56). Toisin sanoen, pyritään toteamaan, että ratkaisu täyttää käyttäjävaatimukset maksimikuormituksen osalta.

4.8 Savutestaus

Savutestauksella tarkoitetaan yksinkertaisia perustestejä, kuten "käynnistyykö ohjelma pikakuvakkeesta", joilla pyritään osoittamaan, ettei järjestelmään ole syntynyt karkeita puutteita tai ilmeisiä virheitä. (Kasurinen 2013, 56–57).

4.9 Ad hoc testaus

Ad hoc testaus nimensä mukaan on täysin suunnittelematonta ja dokumentoimatonta testausta, jota yleensä sovelluksen kehittäjä tekee ohjelmointityön ohessa. Ad hoc testauksesta ei kirjata ylös mitään yksityiskohtia tai varsinaisesti suunnitella mitään etukäteen. Yleisesti, ohjelman kehittäjä vain testaa, että kaikki näyttäisi toimivan. (Kasurinen 2013, 60)

4.10 Tutkiva testaaminen

Tutkiva testaaminen perustuu testaajan ammattitaitoon ja ymmärrykseen siitä missä virheitä saattaa olla. Tutkivassa testauksessa, suunnittelu, toteutus ja järjestelmään tutustuminen tehdään samanaikaisesti luottaen testaajan kykyihin. Tutkiva testaaminen eroaa ad hoc testaamisesta sillä, että se voi perustua olemassa oleviin malleihin ja tunnistettuihin riskeihin. Tutkivalla testaamisella voidaan löytää uusia testitapauksia tuleviin ohjelmistoversioihin ja havaita ongelmia, jotka eivät ole niin ilmeisiä. (Kasurinen 2013, 59).

4.11 Mallipohjainen testaaminen

Mallipohjainen testaaminen suoritetaan ohjelmistosta esisuunnitteluvaiheessa luotujen mallien perusteella. Tämä lähestymistapa vaatii, että ohjelmistosta on luotu riittävän tarkat mallit, kuten UML-kuvaukset (Unified Modeling Language). Mallipohjaisen testaamisen etuna on se, että toimivat testitapaukset osoittavat myös sen, että kehitetty ohjelma toimii niin kuin mallinnettu. (Kasurinen 2013, 59–60).

4.12 Alfa- ja betatestaus

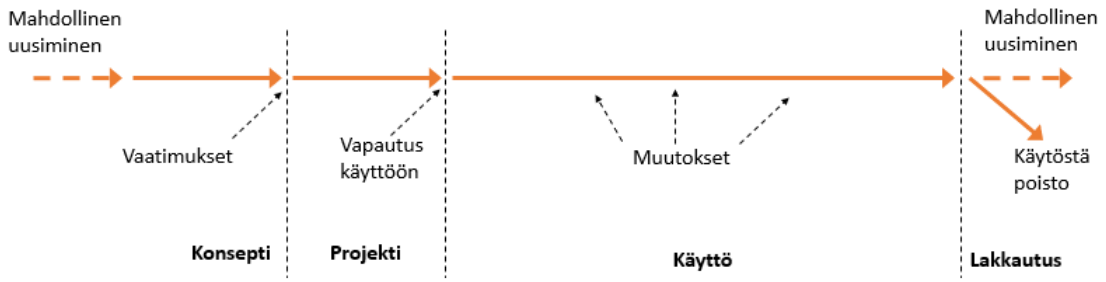
Alfa- ja betatestaus eivät ole varsinaisia testausmenetelmiä, vaan enemmänkin testausstrategiaan liittyviä työvaiheita, jossa selvitetään toimivuutta kokonaisuutena tuotteen kehityselinkaaren loppuvaiheessa, ennen lopullista julkaisua. Alfa- ja beta-testaukset termeinä liittyvät yleensä ns. OTS-tuotteisiin (Off-the-shelf), kuten tietokonepelit. Monet ohjelmistokehitysmallit, kuten V-malli, eivät näitä termejä tunne. (Kasurinen 2013, 58).

5. TIETOKONEISTETTUJEN JÄRJESTELMIEN HANKINTA

Tässä kappaleessa perehdytään ISPE GAMP 5 ja viranomaisohjeistuksen näkökulmista tyypillisiin tietokoneistettujen järjestelmien hankintaan liittyviin elinkaarimalleihin lääketeollisuudessa. Elinkaarimallit sisältävät hankinnan, validointitestauksen ja projektihallinnan eri toimenpiteitä, laajuutta, rooleja, sekä laaturiskienhallinnan näkökohtia kriittisen ajattelun tukemana. Toisin sanoen, kappaleen sisältö edustaa perinteistä lääketeollisuuden ohjelmistojen hankintaelinkaarta, jonka piirteet ja ominaisuudet tulisi esiintyä myös ketterässä kehitysmallissa (ks. kappale 3.5 Ketterä kehitys lääketeollisuudessa).

5.1 Tietokoneistetun järjestelmän elinkaarimalli

GAMP 5 Ohjeistaa lääketeollisia toimijoita lähestymään tietokoneistettujen järjestelmien hankintaa elinkaariajattelun avulla, joka sisältää neljä päävaihetta, konsepti, projekti, käyttö ja lakkautus (ISPE 2nd 2022, 24), joka noudattaa ASTM E2500 standardia (ISPE 2nd 2022, 23). Seuraavassa kuvassa (Kuva 6) neljä päävaihetta sijoitettuna kuvitteelliselle aikajanalle keskeisten tavoitteiden kanssa.



Kuva 6. *Elinkaaren neljä päävaihetta (mukaillen, ISPE 2nd 2022, 24).*

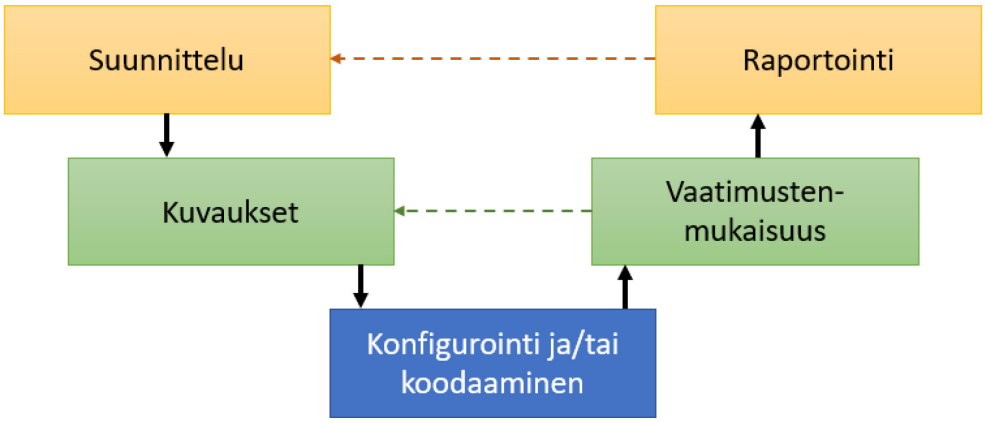
Konseptivaiheessa yritys arvio mahdollisuuksia automatisoida yhden tai useamman liiketoimintaprosessin. Tyypillisesti tässä vaiheessa asetetaan toteutuksen laajuus, lähtövaatimukset ja arvioidaan mahdolliset toimittajat. Konseptivaiheesta (esisuunnittelu) siirrytään projektivaiheeseen, kun toteutuksen laajuus, kustannukset ja hyödyt ovat toteutuksen kannalta suotuisat. Projektin päättyessä järjestelmä otetaan käyttöön, kunnes lakkautetaan tai uusitaan. Käytön aikana järjestelmän vaatimustenmukaisuutta ylläpidetään ja arvioidaan, sekä lakkautuksen yhteydessä järjestelmään tallennettujen

tietojen kohdalla arvioidaan niiden säilyttämisestä (arkistointi) tai siirtämisestä korvaavaan järjestelmään. (ISPE 2nd, 24.)

GAMP 5 elinkaarimalli ei sinänsä esitä mitään vaihetta projektin lakkauttamiseen tai ylipäätään keskeytymiseen jo konseptivaiheessa (Kuva 6), mutta ovat toki myös vaihtoehtoisia elinkaaripolkuja. Tämän työn kannalta olennainen vaihe on projekti, tai järjestelmän käytön aikainen muutos.

5.2 Validoidun tietojärjestelmän elinkaarimalli

Ohjelmistokehityksen elinkaarimalli (Systems Development Life Cycle, SDLC) on kuvaus tai ohjeellinen kuvaus siitä, miten ohjelmisto on kehitetty tai pitäisi kehittää (Marciniak 2001, 3). GAMP 5 kuvaa perinteiseksi lähestymistavaksi ohjelmiston elinkaarimalleiksi spesifikaatioverifikaatiomallia (Kuva 7) ja tämän lisäksi uudessa painoksessa vaihtoehtoisesti myös iteratiivisen ja inkrementaalisen mallin (ISPE 2nd 2022, 25), jota ei siis aiemmassa GAMP 5 versiossa ollut. GAMP 5 spesifikaatioverifikaatiomalli vastaa käytännössä SDLC V-mallia (W3schools® of Technology 2022) vahvalla riskienhallintanäkökulmalla lisättynä. Kasurinen (Kasurinen 2013, 38) kuvaa samalla mallilla testaukseen liittyviä tasoja, mutta lähinnä GAMP 5 määrittelemän räätälöityjen sovellustuotteiden mukaisesti (ks. Kuva 11). Tähän lienee syynä se, että Kasurisen aihe käsittelee ohjelmistotuotantoa, kun vastaavasti GAMP 5 ottaa huomioon myös valmiiden ohjelmistotuotteiden käyttöönoton, ns. OTS-tuotteet (Off The Shelf).



Kuva 7. *GAMP 5 elinkaarimallin projektivaiheet yleisesti (mukaillen ISPE 2nd 2022, 30).*

GAMP 5 kuitenkin toteaa, että se kuvaa elinkaarimalleja vain kokonaisvaltaisesti, eikä pyri kuvaamaan ohjelmistokehityksen prosesseja yksityiskohtaisesti, joten muut mallin ja lähestymistavat ovat aivan yhtä hyväksyttäviä (ISPE 2nd 2022, 26). GAMP 5 elinkaarimalli kuvaa projektivaiheen sisältävän eri työvaiheita, jotka ovat, suunnitteluvaihe,

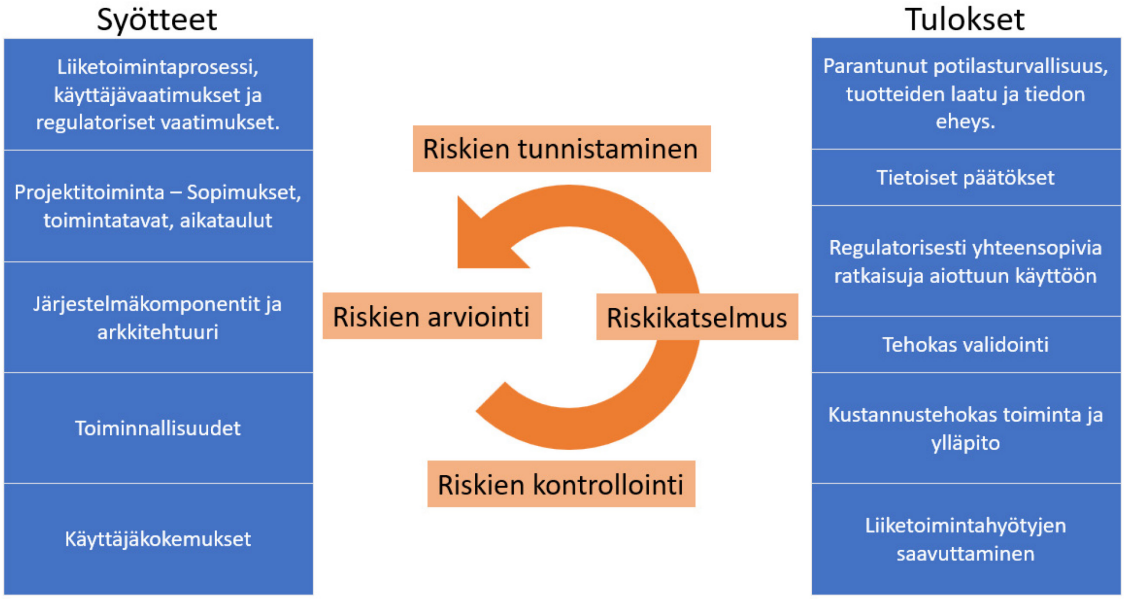
spesifikaatio-, konfiguraatio- ja ohjelmointivaihe, verifiointivaihe ja raportointi- ja julkaisuvaihe, sekä lopulta vapautus käyttöön (ISPE 2nd 2022, 29).

5.3 Laaturiskien hallinta

EMA:n julkaisema ICH ohje Q9 (ICH Q9) sisältää periaatteita ja esimerkkejä mahdollisista laaturiskien hallinnan työkaluista (QMR), joita voi soveltaa farmaseuttiseen laadun eri näkökohtiin, kuten valmistus. Q9 määrittelee kaksi perusperiaatetta laaturiskien hallinnalle:

- Laatuun kohdistuvien riskien arviointien on perustuttava tieteelliseen tietoon ja ennen kaikkea yhdistyä potilaan suojeluun.
- Muodollisuuksien, työmäärien ja dokumentaatioiden pitää olla oikeassa suhteessa riskin tasoon nähden. (ICH Q9, 3–4.)

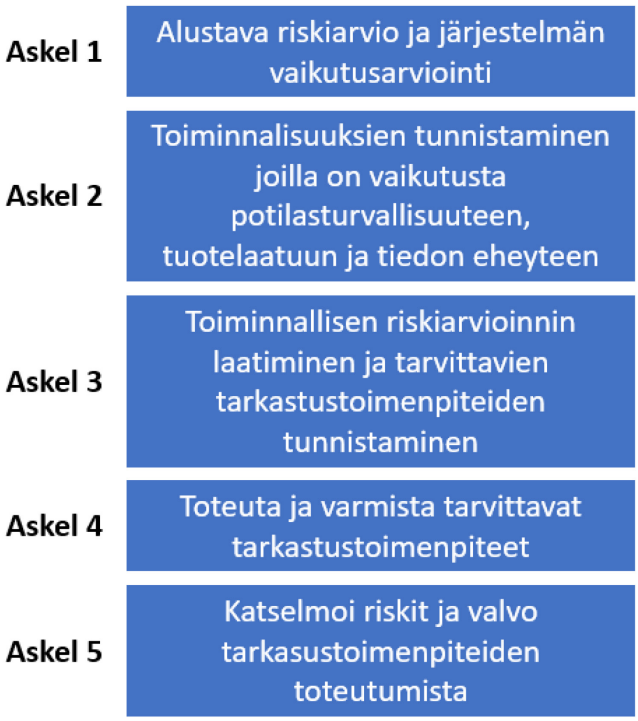
GAMP 5 kuvaa QMR prosessia iteratiiviseksi, joka kattaa koko tietokoneistetun järjestelmän elinkaaren konseptista lakkautukseen asti ja se toimii parhaiten, kun se on liitetty kiinteäksi osaksi yrityksen kokonaisvaltaista laatujärjestelmää (ISPE 2nd 2022, 49). Tietokoneistettujen järjestelmien osalta tieteellinen tieto pohjautuu järjestelmän spesifikaatioihin ja tietoon järjestelmän tukemasta liiketoimintaprosessista (ISPE 2nd 2022, 51). Kuvassa alla (Kuva 8) on havainnollistettu iteratiivisen QRM prosessin vaiheet, syötteet ja tulokset.



Kuva 8. *QRM prosessin vaiheet, syötteet ja tulokset (mukaillen ISPE 2nd 2022, 49).*

Riskejä voidaan hallita suunnitteluratkaisuilla tai vähentämällä riskiä hyväksyttävään tasoon asti muilla keinoilla, kuten ohjeistamalla, tai testaamisella osoitetaan, että riskit

ovat hyväksyttävien rajojen puitteissa. Suositelluin tapa vähentää riskejä on joko muokkaamalla toimintaprosessia tai tietokoneistetun järjestelmän toteutusta. Säännöllisillä toteutuskatselmuksilla voidaan vaikuttaa riskeihin jo projektin aikana. (ISPE 2nd 2022, 50.) GAMP 5 oppaan mukainen laaturiskien hallintaprosessi havainnollistettu seuraavassa kuvassa (Kuva 9).



Kuva 9. *GAMP 5 oppaan kuvaama laaturiskien hallintaprosessi (mukaillen ISPE 2nd 2022, 51).*

Ennen varsinaiseen projektivaiheeseen siirtymistä tulee tehdä alustava riskiarviointi, joka perustuu liiketoimintaprosessien ja liiketoimintariskien ymmärrykseen, alustaviin käyttäjävaatimuksiin, sääntelyvaatimuksiin ja muihin tunnettuihin toiminnallisiin vaatimuksiin, kuten esimerkiksi koneturvallisuus- ja työturvallisuusvaatimukset. Kaikki aiheeseen liittyvät aiemmat arviot voivat olla hyödyllisiä, joten niitä ei tule toistaa tarpeettomasti. Alustavan arvion tulee sisältää päätös siitä, että onko järjestelmä säännellyn toiminnan piirissä. (ISPE 2nd 2022, 89.) Säännöllisten tarkastusaktiviteettien tiheyden ja laajuuden tulee perustua riskitasoon ja ottaa huomioon aikaisemmat löydökset sekä toimintaan liittyvä kokemus ja historia (ISPE 2nd 2022, 54).

5.4 Laaturiskien tunnistaminen

Kriittisellä ajattelulla tarkoitetaan kykyä analysoida tietoa objektiivisesti ja tehdä perusteltu arvio. Siihen kuuluu lähteiden, kuten tietojen, tosiasioiden, havaittavien ilmiöiden

ja tutkimustulosten arviointi. Kriittisesti ajattelevat voivat tehdä perusteltuja johtopäätöksiä tiedoista ja erottaa hyödylliset ja vähemmän hyödylliset yksityiskohdat toisistaan ratkaistakseen ongelmia ja tehdäkseen johtopäätöksiä. (Doyle, 2022).

Kriittistä ajattelua tulisi soveltaa kokonaisvaltaisesti koko liiketoimintaprosessiin, jota tietokoneistettu järjestelmä tukee. Tukijärjestelmien ja liitäntärajapintojen rooli olisi sisällytettävä toteutusprojektin, validoinnin ja operatiivisten toimien piiriin, jotta vältetään mahdollisten potilasturvallisuuteen, lääketuotteiden laatuun ja tietojen eheyteen tunnistettavien mahdollisten riskien puuttuminen. (ISPE 2nd 2022, 177.)

Jotta kriittistä ajattelua voidaan käyttää menestyksekkäästi, laadunhallintajärjestelmän ja validointipolitiikan on kannustettava sen soveltamiseksi. Esimerkkejä tästä ovat:

- Jos tarjolla on lomakemalleja, niiden rooli on toimia apumuistina, eikä pelkästään täytettävinä lomakkeina, jotta järjestelmän osalta varmistetaan tietyn asian kannalta merkitykselliset aiheet.

- Tämä lisäksi tai vaihtoehtona lomakemalleille voidaan tarjota olennaiset aihealueet, jotka on otettava huomioon tai on oltava osana aktiviteetteja, jolloin muoto ja rakenne jätetään avoimeksi työmäärän optimointia varten.

Kriittistä ajattelua tukee erilaiset liiketoimintaprosessia ja tietoliikennettä kuvaavat vuokaaviot ja toimintakuvaukset, sekä niihin liittyvien aliprosessien toimintakuvaukset ja kaaviot. (ISPE 2nd 2022, 172.)

5.5 Validointistrategia

Validointistrategialla pyritään siihen, että hankittava järjestelmä on vaatimusten mukainen ja se, että soveltuu aiottuun käyttöön. Strategian laadinta perustuu riskiarvioon, järjestelmäkomponenttien ja arkkitehtuurin arviointiin, sekä toimittajan arviointiin. Kaikkien suoritettujen arviointien keskeiset päätelmät tulee sisällyttää strategiaan, sekä tämän lisäksi kaikki muut erityiset menettelyt ja standardit. Validointistrategian tulee selostaa elinkaarimalli ja spesifikaatioverifikaatiomalli sisältäen kehitystapamallin, kuten ketterä kehitysmalli. Tämä lisäksi strategiasta pitää selvitä projektin jokaisessa vaiheessa tarvittavat sisällöt ja tulokset, sekä hyväksymiskriteerit. Strategian tulee kuvata lähestymistavat jäljitettävyydelle, suunnittelutarkastuksille ja toimittajatoimintojen hyödyntämiselle asianmukaisten varmistusmekanismien kanssa. (ISPE 2nd 2022, 89.)

Validointistrategia on käytännössä suunnitelma siitä, mitä laatuprosesseja ja todisteita toimitettavan järjestelmän vaatimustenmukaisuuden varmistamiseksi tarvitaan. Tähän

vaikuttavat toimitettavan järjestelmän elinkaarimalli (kappale 5.2), kompleksisuus (Taulukko 2), toimintaan liittyvät tunnistetut riskit (kappale 5.4), sekä arvio toimittajan kyvystä toteuttaa ratkaisu. Varmistusmekanismit ovat käytännössä eri asiantuntijoiden suorittamia tarkastus- ja hyväksymiskäytäntöjä.

Toimittajan kyky arvioidaan erillisellä arviointiprosessilla, josta GAMP 5 oppaassa on oma liitteensä (ISPE 2nd 2022, 93–106). Liitteestä lyhyesti, säänneltyjen yrityksien tulee huolehtia dokumentoidusti siitä, että valittujen toimittajien ja yhteystyökumppanien palvelujen laatu ja luotettavuus on sääntelyn vaatimalla tasolla. Tässä yhteydessä dokumentointi tarkoittaa mm. laatusopimusta toimittajan kanssa.

5.6 Validointimallit

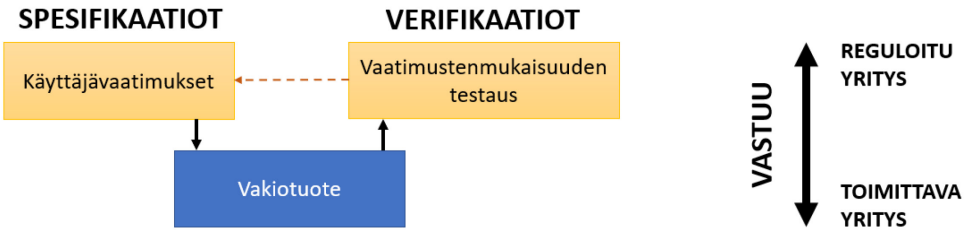
ISPE (ISPE 2nd 2022, 33–37) ohjeistaa projektitoteutuksia eri tasoisilla V-mallin prosessimalleilla riippuen validoitavan sovelluksen kompleksisuudesta. Sovelluksen kompleksisuus voidaan määrittää ISPE:n määrittämien neljän eri kategorioiden mukaan (ISPE 2nd 2022, 131–132), näistä mukailtu taulukko alla (Taulukko 2) esimerkkien kanssa.

Kategoria	Selitys	Esimerkit
1. Infrastruktuuri	Ohjelmistot, jolla ylläpidetään toimintaympäristöä.	mm. Käyttöjärjestelmät, palomuurit, virustorjuntaohjelmistot, tietokantaohjelmistot, erilaiset monitorointityökalut.
3. Standardiohjelmisto	Ohjelmistot, joissa voidaan säätää joitakin ajonaikaisia parametrejä, mutta se ei ole konfiguroitavissa liiketoimintaprosessien mukaiseksi.	Laiteohjelmistot ja valmisohjelmistot, kuten erilaiset toimistosovellukset ja työkaluohjelmistot.
4. Konfiguroitava ohjelmisto	Ohjelmistot, jotka ovat konfiguroitavissa liiketoimintaprosessille sopivaksi lähdekoodiin koskematta.	Laboratorio- ja ohjausjärjestelmäsovellukset. Kiinteistöautomaatiojärjestelmät, yksinkertaiset käyttöliittymäsovellukset. Nämä esimerkit voivat sisältää kategoriaan 5 kuuluvia ohjelmistomoduuleja.
5. Räätälöity ohjelmisto tai ohjelmistokomponentti	Ohjelmistot, jotka ovat liiketoimintatarkoitukseen kehitetty ja ohjelmoitu.	Kuten kategoria 4, mutta sisältää yrityksen itsensä kehittämää tai ulkoiselta toimittajalta tarkoitukseen kehitettyä lähdekoodia.

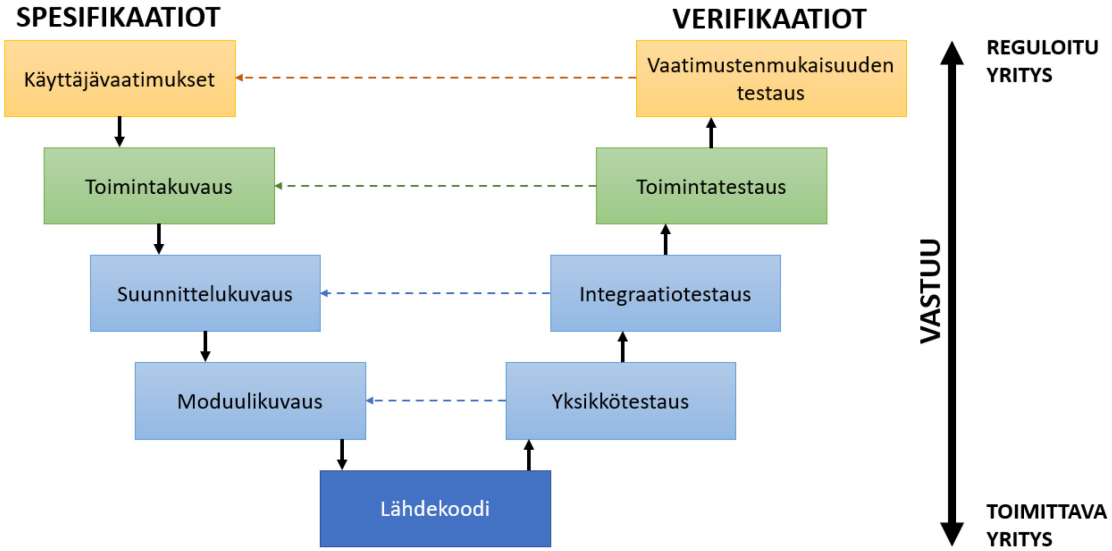
Taulukko 2. *ISPE GAMP5:n mukaiset ohjelmistokategoriat, huomaa että kategoriasta 2 on luovuttu (mukaillen, ISPE 2nd 2022, 131–132).*

Kategoriasta 2 on luovuttu, ja sillä tarkoitettiin laiteohjelmistoja (firmware), koska sen oletetaan olevan kiinteä osa infrastruktuuria.

Lähtökohtaisesti pyritään siihen, että mitä kompleksisemmasta ratkaisusta on kyse, sitä enemmän testaamiseen tulee kiinnittää huomiota. Tämä havainto voidaan tehdä myös tarkastelemalla GAMP 5 ohjeistamia lähestymistapoja kategorioiden 3 ja 5 välillä (Kuva 10, Kuva 11).



Kuva 10. *Validointilähestymistapa vakiosovellustuotteille kategoriassa 3 (mukaillen ISPE 2nd 2022, 34).*



Kuva 11. *Validointilähestymistapa räätälöidyille sovellustuotteille kategoriassa 5 (mukaillen ISPE 2nd 2022, 37).*

Säänneltyjen yritysten on päätettävä vaadituista määrittely- ja varmistustasoista. Näiden lisäksi monia hankkeen vaiheita ja suoritteita voidaan delegoida toimittavalle yritykselle. Jos järjestelmä on uusi, kattava testaus tulisi suorittaa toiminnallisella tasolla, mutta myös suunnittelutasolla.

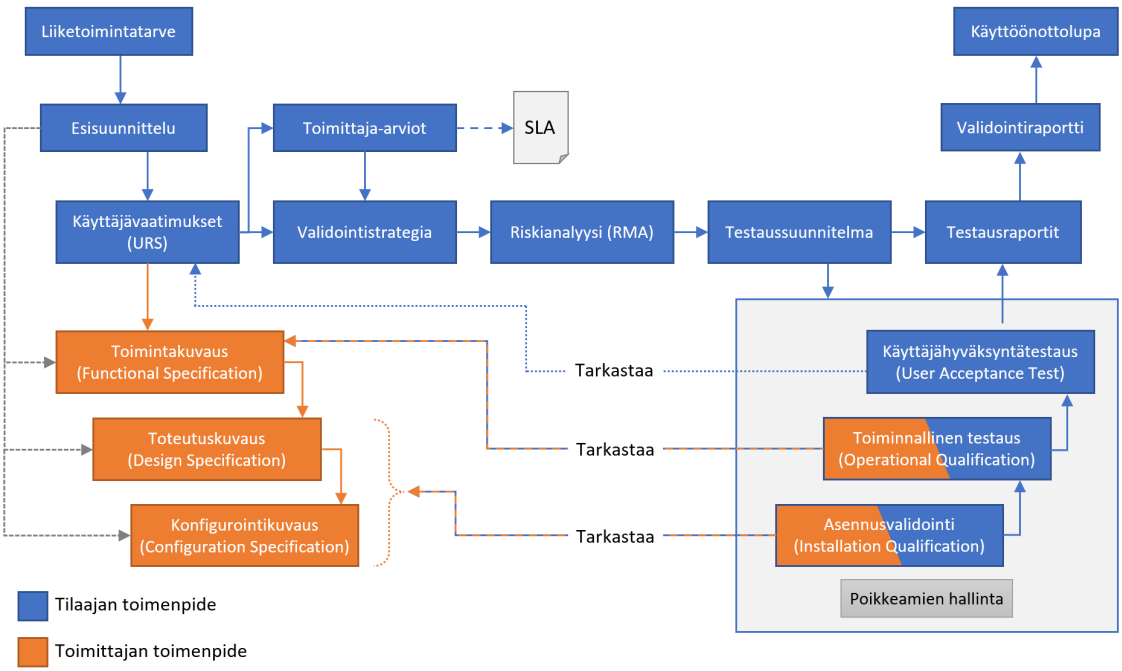
Testaus tyypillisesti kattaa:

- Oikean asennuksen
- Sovelluksen toiminnallisuuden ja suunnittelun
- Testaukset, jotka osoittavat soveltuvuuden aiottuun käyttöön, sekä hyväksyy järjestelmän toteutuksen käyttäjävaatimuksia vasten
- Riskiarvioinnissa ja toimittaja-arvioinneissa tunnistetut lisätestaukset.

Toimittajatoimintaan kuuluu tyypillisesti spesifikaatioiden ja testausspesifikaatioiden tuottaminen säännellyn toimijan puolesta, uusien ohjelmistojen kehittäminen, testaus, käyttäjädokumentaatio, koulutus sekä tuki- ja ylläpitotoiminta. Monimutkaiset järjestelmät voivat vaatia enemmän kuvaushierarkiaa (spesifikaatioita), joka kattaa lisäksi mm. laitteiston suunnittelukuvaukset ja kokoonpanotiedot. (ISPE 2nd 2022, 37.) Säännellyn toiminnan piirissä olevan yrityksen tietokoneistettujen järjestelmien validointi perustuu tunnistettujen riskeihin ja niiden hallintaan, kuten asia kappaleessa 2.3 tulee esille.

5.7 Validointiprosessi

Validointiprosessin keskeinen osa on testaaminen, jolla varmistetaan, että tietokoneistettu järjestelmä soveltuu aiottuun käyttöön. Prosessin tarkoituksena on varmistaa, että järjestelmän valinta, toteutus, konfigurointi ja sen käyttäminen käyttöympäristössä muodostavat yhdessä tietokoneistetun järjestelmän, joka pystyy tukemaan liiketoimintaprosessia (ISPE 2nd 2022, 215). Validointiprosessi on sama riippumatta toteutuksen laajuudesta mutta testauksen ja dokumentoinnin laajuus voi vaihdella, esimerkiksi erilaisten dokumenttien määrässä ja tarpeessa. Kuvassa (Kuva 12) on kuvattu validointiprosessi kokonaisuudessaan, joka käynnistyy liiketoimintatarpeesta päättyen käyttöönottolupaan.



Kuva 12. *Validointiprosessi keskeiset vaiheet (koostettu ISPE 2nd 2022, 216–217)*

Varsinainen validointiprosessi käynnistyy, kun käyttäjäorganisaation on laatinut käyttäjävaatimuskuvauksen tai spesifikaation (URS), ja se on hyväksytty prosessiomistajan

sekä laatuorganisaation toimesta. Käyttäjävaatimuksien laadintaan osallistuu käyttäjäorganisaation lisäksi myös muita sidosryhmiä, kuten validoinnista vastuussa oleva henkilöt. Tällöin alustavan validointistrategian ja riskianalyysin laatiminen voidaan aloittaa mahdollisimman aikaisessa vaiheessa. Validointiprosessi sisältää tilaajan, eli sääntelyn piirissä olevan yrityksen toimenpiteitä, sekä toimittajan tekemiä toimenpiteitä. Tilaajan kanssa sovittu palvelusopimus (SLA) kattaa laadunhallinnalliset vaatimukset, sekä tilaajan vastuut ja rajaukset kokonaistoimituksessa. (ISPE 2nd 2022, 215–217.)

Validointiprosessi sisältää erilaisia kuvausdokumentteja tai spesifikaatioita, joita vasten laaditaan vastaavia testaustapauksia (Kuva 12), kuten spesifikaatioverifikaatiomalli edellyttää. Seuraavassa kuvassa on esitettynä edellä esitetyn validointiprosessin vaiheet loogisessa järjestyksessä kuvitteellisella aikajanalla (Kuva 13).

Kuva 13. *Validointiprosessin vaiheet yksiulotteisesti.*

Yllä esitetty validointiprosessi käytännössä vaatii vesiputousmallista lähestymistapaa toteutuksen osalta, koska riskienhallintaprosessi edellyttää kuvauksia toteutettavista ratkaisuista, joiden perusteella varsinaiset riskit pyritään tunnistamaan. Tämä lähestymistapa ei tue ketterää kehitysmallia, joten validointiprosessin periaatteita pitää soveltaa jollakin toisella tavalla. Käytännössä se voisi tarkoittaa, että ketterässä kehitysmallissa suoritetaan validointiprosessi jokaisella iteraatiokierroksella, mutta se vaatinee ketterämpää toimintatapaa ollakseen tehokas.

5.8 Roolit

Yhtenä GMP vaatimuksena on tunnistaa tarvittavat roolit tietokoneistettujen järjestelmiin liittyen. Minimissään pitää olla nimettynä prosessin omistaja (Process owner), sekä järjestelmän omistaja (System owner), sekä käyttäjät (ks. 2.5 Tietokoneistetulta järjestelmältä vaadittavat ominaisuudet). Muutoshallinnan ja hankinnan yhteydessä edellä mainittujen roolien lisäksi tarvitaan mm. validointi-insinööriä, laadunvarmistusta, sekä muita teknisiä asiantuntijoita aihealueen ja teknisen vaativuuden mukaan (ISPE 2nd. 2022, 60–64). Nämä roolit niihin liittyvät vastuut tulee kirjata validointisuunnitelmaan (ISPE 2nd. 2022, 58).

6. VALIDOINTITESTAUS

Tässä luvussa perehdytään testaustoiminnan automatisoinnin hyödyntämiseksi lääketeollisuuden validointitestaukseen, sen laajuuteen ja käsitteisiin lähinnä viranomaisohjeisiin, ISPE GAMP 5 ohjeistukseen. Lisäksi tässä kappaleessa tutustutaan testauskäsitteisiin GAMP 5 ohjeistuksen näkökulmista tarkasteltuna. GAMP 5 oppaan mainitsemat muut testauskäsitteet sivuavat yleisiä sovellusteollisuuden testauskäsitteitä, joita käsiteltiin aiemmassa kappaleessa (ks. 4 Testausautomaatio ohjelmistokehityksen tukena).

6.1 Testaussuunnittelu

Testaussuunnittelulla tähdätään siihen, että varmistetaan riittävä testauskattavuus tietokoneistetun järjestelmän GxP-toiminnoista perustuen yrityksen riskinsietokykyyn (ISPE 2nd 2022, 218). Validointitestauksella ei varmisteta varsinaisten tuotantoprosessien soveltuvuutta aiottuun käyttöön. Tätä varten suoritetaan kattavasti kvalifiointitoimenpiteitä, jotka sisältävät hyvin samankaltaisia vaiheita kuin validointi, mutta kvalifiointitoimenpiteet kohdistuvat fyysisiin ympäristöihin ja ominaisuuksiin, kuten puhdastilat, tuotantokoneet ja -laitteet sekä varsinaiset tuotantoprosessit (EudraLex Volume 4 Good Manufacturing Practice Annex 15 2015, 4).

Validoinnin riskiperusteisen lähestymistavan luonteeseen kuuluu, että kaikkia toimintoja ei voida kyseenalaistaa ja näin ollen kaikkia mahdollisia vikoja ei löydetä. Kuitenkin kattavasti suoritettujen riskienhallinnallisten toimenpiteiden pitäisi varmistaa, että mahdollisten havaitsemattomien vikojen vaikutukset potilasturvallisuuteen, lääketuotteiden laatuun ja tallennettujen tietojen eheyteen ovat vähäisiä. Validointitestauksen kattavuus tulisi rajoittaa tiukasti aiottuun käyttöön kohdistuen, sekä testausdokumentaation laajuus tulisi mitoittaa tunnistettujen riskien laajuuden mukaan. (ISPE 2nd 2022, 218.)

Testaussuunnitelmat usein koostuvat yhdistelmästä toimittajan tekemiä testejä (OQ), käsikirjoittamattomia ja käsikirjoitettuja testejä, sekä lopuksi varsinaisista käyttäjähyväksyntätestejä (UAT). Muutoksien yhteydessä, muutoshallinnan välineenä käsikirjoittamattomia ja käsikirjoitettuja testitapauksia voidaan toistaa, jotta voidaan todeta, että tietokoneistettu järjestelmä soveltuu muutoksen jälkeenkin aiottuun käyttöön. (ISPE 2nd 2022, 218.)

Käytännössä tilaajan validointitoimenpiteet tapahtuvat kvalifiointitoimenpiteiden rinnalla niiltä osin missä tietokoneistettu järjestelmä tukee tuotantoprosessia, ts. validointitestauksen keskeinen rooli on tähdätä siihen, että riskienhallintanäkökulmista tarkasteltuna, tuotantoprosesseja tukevat tietokoneistetut järjestelmät toimivat aiotulla tavalla oikeassa toimintaympäristössä.

6.1.1 Testitapausten jäljitettävyys vaatimuksiin

Viranomaisohjeet säätävät, että käyttäjävaatimuksien teknisten tietojen tulee kuvata tarvittavat toiminnot tietokoneistetussa järjestelmässä ja vaatimuksien on perustuttava dokumentoituihin riskiarviointeihin ja hyvien tuotantotapojen vaikutuksiin. Käyttäjävaatimukset tulee olla jäljitettävissä toteutuksiin järjestelmän koko elinkaaren ajan. (EudraLex Volume 4 Good Manufacturing Practice: Annex 11 2011, 3.)

ISPE vastaavasti ohjeistaa luomaan järjestelyt testitapauksien liittämiseksi järjestelmää kuvaaviin spesifikaatioihin, sekä asiaankuuluvin menettelyin pitämään asiayhteydet yllä ajantasaisesti. Järjestelyn pitäisi mahdollistaa jäljitys testitapauksesta asiaankuuluvaan spesifikaatioyksityiskohtaan. Jäljitettävyyden tarkkuudella on suora riippuvuus dokumentaation täydellisyyteen ja tarkkuuteen. (ISPE 2nd 2022, 136–137.)

Testaamisen kannalta on olennaista, että jäljitettävyys testitapauksen ja vaatimuksen välillä on selkeä, jotta riskienhallinnan näkökulmasta testien kokonaiskattavuus voidaan todentaa. Tästä syystä eri testikokonaisuuksien välillä voi olla esiintyä päällekkäisyyttä, vaikka todellisuudessa siihen ei ole tarvetta kontrolloiduissa testausympäristöissä. (ISPE 2nd 2022, 218.)

6.2 Toimittajan tekemä testaus

Toimittajan tekemä testaus on toimittajan omaan laatujärjestelmään kuuluva testaaminen, jonka soveltuvuus säädeltyyn toimintaan on vahvistettu toimittaja-arvioinnin yhteydessä. Toimittajan suorittamia testitapauksia ei ole tarpeen toistaa tilaajan toimesta, mikäli siihen ei nähdä riskienhallinnan kannalta erityistä tarvetta. Tämä kuitenkin edellyttää, että toimittajan tekemät testit ovat tilaajan saatavilla ja tarkastettavissa, sekä sen, että testitapaukset ovat jäljitettävissä tilaajan vaatimuksiin. (ISPE 2nd 2022, 228–229.)

6.3 Käsikirjoittamaton testaus

Käsikirjoittamattomat testaukset ovat testaamistapoja, jotka ovat luokiteltavissa neljään alakategoriaan, ad-hoc, virheen arvaus, tutkiva ja päivä elämässä testaus. Kaikkien

näiden neljän kategorian testaustuloksiin kirjataan yksityiskohtainen kuvaus testatuista vaatimuksista tai toiminnallisuuksista, mahdollinen ongelma, tulokset ja/tai johtopäätökset, sekä kuka suoritti testauksen ja milloin (päivämäärä). (ISPE 2nd 2022, 218–221.)

Ad-hoc testaaminen tapahtuu ilman testaussuunnitelmaa. Virheen arvaus testaaminen perustuu testaamiseen missä odotetaan ennalta määritettyjä tai arvattavia virhetilanteita. Tutkivassa testaamisessa testaaminen perustuu korkeamman tason tavoitteiden testaamiseen, kuten jonkin prosessin vieminen alusta loppuun ilman askel askeleelta opastamista, tällöin tulokset kirjataan tavoitetaso läpäisty tai epäonnistuttu. Päivä elämässä testaaminen on samankaltainen tutkivan testaamisen kanssa, mutta siinä testitapauksien tavoitteet asetellaan liiketoimintaprosessin tuntemukseen perustuen tai toimintaohjeita vasten. (ISPE 2nd 2022, 218–221.)

6.4 Käsikirjoitettu testaus

Käsikirjoitettu testaamien jakaantuu kahteen alakategoriaan, rajoitettu (limited) ja vankka (robust). Käsikirjoitetut testit eroavat käsikirjoittamattomista sillä, että ne ovat tiukasti ohjattuja askel askeleelta. Käsikirjoitetut testitapaukset katselmoidaan ja hyväksytään ennalta ja samalla tavalla toimitaan suoritettujen testien osalta. Rajoitetun ja vankan testitapauksen ero on, että rajoitetussa testitapauksessa suoritetaan yksittäisiä testitapauksia ja vastaavasti vankassa testauksessa suoritetaan laajempi toiminnallisuus askel askeleelta. (ISPE 2nd 2022, 218–222.) Käsikirjoitettuja testitapauksia tulee käyttää toimintoihin, joille on tunnistettu korkea riski. Käsikirjoitettuja testitapauksia voidaan toistaa regressiotestaustarpeissa, kuten esimerkiksi muutoksien yhteydessä. Testausautomaatiolla toteutetut testitapaukset luokitellaan käsikirjoitetuiksi testitapauksiksi. (ISPE 2nd 2022, 223.) Omiin testauskokemuksiini perustuen, käsikirjoitetut testit noudattavat kappaleessa 4 esitettyjä määritelmiä, tyypillisesti mustalaatikkotestausta.

6.5 Käyttäjähyväksyntätestaus

Käyttäjähyväksymistestaus (UAT) ei ole varsinainen testaustapa, vaan sillä tarkoitetaan kaikkia edellä mainittuja testaustapojen kokonaisuutta, jolla osoitetaan tietokoneistetun järjestelmän soveltuvuus aiottuun käyttöön (ISPE 2nd 2022, 218). Toisin sanoen, UAT testauksella katetaan kaikki sellaiset testitapaukset, joita ei ole aiemmissa vaiheissa testattu, voitu testata tai riskienhallinnallisten näkökulmien takia halutaan testaamalla varmistua toteutuksen laadun yksityiskohdista.

6.6 Testausautomaatiovaatimukset

GAMP 5 nostaa esille, että automaattisen testauksen työkaluilla on tärkeä rooli koska ne auttavat varmistamaan perusteellisen testauksen, joka on usein automatisoitu ja sisältää regressio- ja raja-/stressitestiominaisuuksia. Tällaiset työkalut myös usein tallentavat niihin liittyvät testitulokset. Lähtökohtaisesti tulkitaan, että työkaluohjelmistot kuuluvat kategoriaan 1 (ks. Taulukko 2), koska näiden ohjelmistojen ei tulkita kuuluvan GMP säädeltyihin liiketoimintaprosesseihin tai suoraan vaikuttavan elektronisten tallenteiden syntyyn, jotka tukevat lääkinnällisen tuotteen elinkaarta. Tästä syystä työkaluohjelmistot eivät vaadi tietokoneistettujen järjestelmien validointia, mutta ne vaativat kuitenkin asianmukaisen riskiarvioinnin ja perustietotekniikkakäytäntöjen hyvää tukea, kuten ITIL (Information Technology Infrastructure Library) käytännöt. Tämän lisäksi säädellyn yrityksen tulee luetteloida käytetyt työkaluohjelmistot, versiot ja niiden käyttökohteet. Työkalujen valintaan liittyvät avainkriteerit ovat työkalun sopivuus elinkaariprosessin automatisoimiseksi, työkalun vastaavuus asiantuntijoiden käyttötarpeisiin ja työkalun soveltuvuus ylipäätään aiottuun käyttötarkoituksiin. Jos toimittajan ja säännellyn yrityksen on tarkoitus jakaa työkaluja kolmansien osapuolin kanssa, on tietoihin pääsyä harkittava järjestelmän käyttöiän aikana, sekä laadittava suunnitelma työkalujen ja tietojen luovutukseen projektin jälkeen. (ISPE 2nd, 253–254.)

GAMP 5 kannustaa kriittiseen ajatteluun, mutta samalla kehottaa hyödyntämään automaattisia testaustyökaluja ja testinhallintatyökaluja laajojen manuaalisten ponnistelujen sijaan ja viittaamaan työkaluissa oleviin testiartefakteihin dokumentaation sijaan (ISPE 2nd, 177). Tämän lisäksi kriittisen ajattelun tulisi huomioida, että vaatimustenhallintatyökalujen ja automaattisten testityökalujen artefaktien muodossa olevilla tiedoilla ja testitodistuksilla on sama asema kuin virallisella käsin tuotetulla dokumentaatiolla (ISPE 2nd, 175).

Automaattisia testaustyökaluja voidaan käyttää myös siirrettyjen tietojen eheyden testaamiseen. Tällöin, kriittistä ajattelua tarvitaan suunniteltaessa, kuinka tiedon eheys ja valmius tietojen siirtoon varmistetaan. Lisäksi arviot siirrettyjen tietojen laadun varmistamiseen, sekä kuinka paljon tietoja on tarkistettava ja miten siirtoprosessin aikana syntyneet virheet voidaan havaita. (ISPE 2nd, 180.)

7. TILAAJAN VALIDOINTITESTAUS KÄYTÄNNÖSSÄ

Tilaaja suorittaa riskiarviointien ja validointistrategian laatimiset, testaussuunnittelut, manuaalisten testauksien suoritukset ja niihin liittyvät hyväksymiskäytännöt, sisältäen sähköiset allekirjoitukset, dokumenttipohjaisesti siihen suunnitellussa tietokoneistetussa järjestelmässä. Järjestelmän toimintalogiikka perustuu samankaltaisiin vaatimuksiin kuin se hoidettaisiin paperilla ja varsinainen prosessi perustuu GAMP 5 ohjeistuksen mukaiseen spesifikaatioverifikaatiomalliin (ks. Kuva 7). (Soinio Hanna, 2023.)

Tämä edellyttää projektin alkuvaiheessa seikkaperäisiä kuvauksia kaikista järjestelmän toiminnoista, sekä siitä, että miten jokainen käyttäjävaatimus toteutuu. Testaaminen pääsääntöisesti suoritetaan hyvin manuaalisesti suorittamalla operaatioita testattavassa ympäristössä ja havainnoimalla järjestelmän reaktioita. Reaktiot dokumentoidaan testipöytäkirjaan ja allekirjoitetaan (Kuva 14). Tarvittaessa testitapaukseen liitetään kuvakaappauksia tai tulosteita testattavasta järjestelmästä. (Soinio Hanna, 2023.)

1.1	Login into PM control with the user ID and AD password. Role: all roles.	Login was successfully performed with the user ID (CWID) and password from operating systems authentication (= Active Directory)	**Reason for Edit:** All roles tested. **New Value:-** Login was successful with CWID: (Supervisor). Login was successful with CWID:	**New Value:-** Pass **Executed By:** 2020-07-28 06:59:18	**Reason for Edit:** All roles tested. **New Value:-** No Comments **Executed By:**

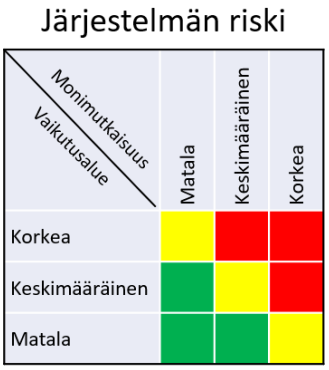
Kuva 14. *Testitapaus suoritetusta testipöytäkirjasta (Soinio Hanna, 2023).*

Tarvittaessa aiemmin hyväksyttyjä testisuunnitelmia voidaan suorittaa uudelleen, esimerkiksi muutoshallintojen yhteydessä, jos esimerkiksi riskiarviossa on päädytty regressiotestaamisen tarpeista jollakin osa-alueella. Testaaminen on usein hidasta ja vaatii eri sidosryhmien ja asiantuntijoiden työpanosta, niin testitapausten laadinnassa kuin suorittamisessa. Testitapauksia sisältäviä dokumentteja voi olla useita kymmeniä testattavaa järjestelmää kohden ja käyttäjähyväksyntätestaustapahtumien kokonaismäärät pyörivät yleensä useissa sadoissa testitapauksissa. (Soinio Hanna, 2023.)

7.1 Tilaajan riskiarvioinnit

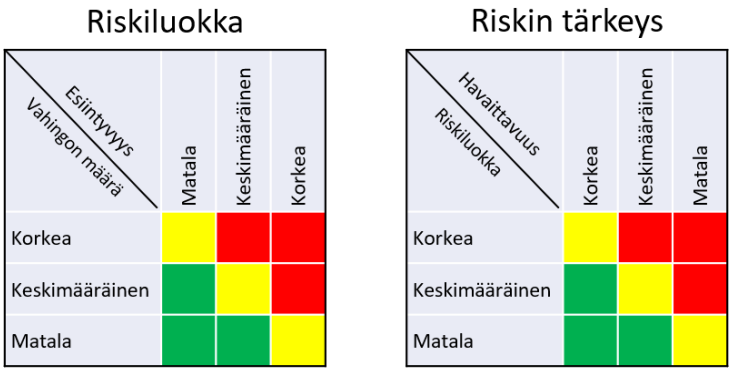
Tilaaja arvioi käyttöönotettavien tietokoneistettujen järjestelmien riskejä kolmesta eri näkökulmasta tarkastellen, jotka ovat järjestelmän kokonaisriskitaso, riskien luokitus, sekä riskien tärkeys. Riskit arvioidaan riskimatriisien avulla siten, että saavat kokonais-

arvioksi, matala, keskimääräinen tai korkea. Järjestelmän kokonaisriski arvioidaan järjestelmän vaikutusalueen ja kompleksisuuden tulona, kuten seuraavassa kuvassa (Kuva 15), jonka tulos vaikuttaa validointistrategiaan (ks. kappale 5.6 Validointimallit) ja järjestelmään tehtäviin säännöllisiin tarkastuksiin käytön aikana. (Soinio Hanna, 2023.)



Kuva 15. *Järjestelmän kokonaisriskin arviointimatriisi (Soinio Hanna, 2023).*

Järjestelmän vaatimuksiin liittyviä riskejä arvioidaan kahden matriisin tulona seuraavan kuvan mukaisesti (Kuva 16), jossa ensimmäisen matriisin tulos, riskiluokka, arvioidaan jälkimmäisessä matriisissa havaittavuuden näkökulmasta. (Soinio Hanna, 2023.)



Kuva 16. *Riskien arviointiin käytettävät arviointimatriisit, huomaa, että ensimmäinen matriisi toimii syötteenä jälkimmäiselle (Soinio Hanna, 2023).*

Lopulta jokaiselle eri vaatimuksille saadaan kokonaisriskitaso, joista punaiselle arvioille päätyneille vaatimuksille tulee laatia suunnitelmat riskin pienentämiseksi. (Soinio Hanna, 2023.)

8. VALIDOITAVA JÄRJESTELMÄINTEGRAATIO OLENNAISILTA OSIN

Tässä kappaleessa esitellään olennaisilta osin tilaajan validoitavan järjestelmäintegraation toteutus informaatiotekniikan (IT) ja operatiivisen tekniikan (OT) välisessä rajapinnassa. Tämän lisäksi pohditaan tilaajan motivaatiotekijöitä integraatioiden rakentamiseksi, sekä arkkitehtuuriratkaisun tuomia etuja ja haittoja testausautomaation hyödyntämisen ja säädellyn liiketoiminnan näkökulmista.

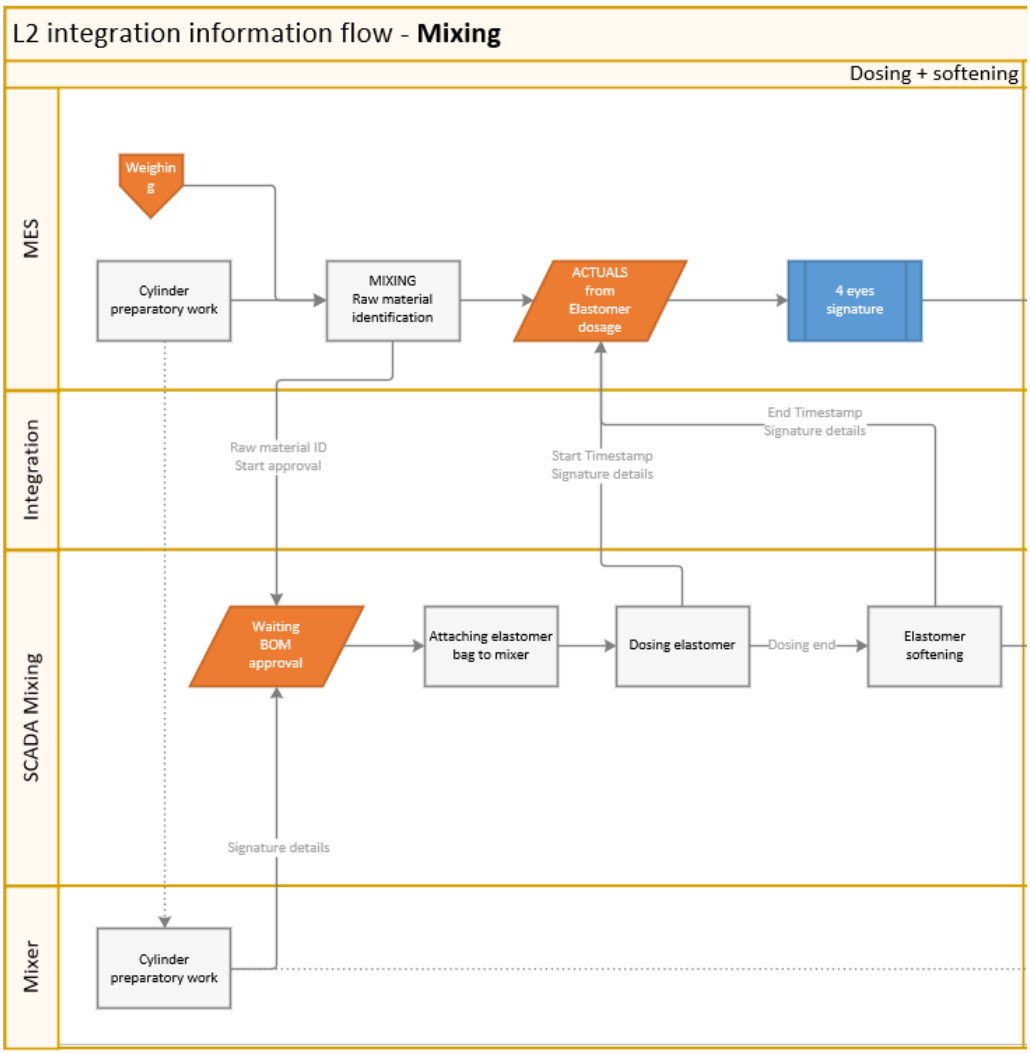
8.1 Käyttäjävaatimukset

Tilaaja oli laatinut tarpeesta käyttäjävaatimusdokumentaation, tilaajan laatujärjestelmän mukaisesti, jossa kuvataan sekä teknisiä reunaehtoja, kuten tietoturvaan liittyviä ehtoja, että varsinaisia sanomia kuvaavia vaatimuksia, josta pieni otanta alla olevassa kuvassa (Kuva 17).

4.3	Transport of the messages	System must have ability to receive and send messages through web service interfaces.	☒	☐	
4.4	Automation level communication	System must have ability to communicate with OPC UA protocol.	☒	☐	
4.5	Transformation of the messages	System must have ability collect, transform, merge, and combine messages from different types and protocols. At least for XML, JSON and OPC UA, but not limited to.	☒	☐	

Kuva 17. *Ote tilaajan käyttäjävaatimusdokumentaatiosta.*

Käyttäjävaatimusdokumentaation liitteenä oli kokonaisprosessia kuvaava uimaratakaavio, johon oli merkitty integroitavien järjestelmien välillä liikkuvien sanomien sijainti aikahorisontissa, sekä niiden suunta ja sisältö. Kuvassa (Kuva 18) nähdään ote käyttäjävaatimuksen liitteenä olevasta uimaratakaaviosta. Laaditusta uimaratakaaviosta nähdään eri hierarkiatasoilla toimivat järjestelmät ja niiden välille tarvittava sanomavaihto ja olennainen tietosisältö ilman tunnistetietoja, kuten eränumero.



Kuva 18. *Ote prosessia kuvaavasta uimaratakaaviosta.*

Uimaratakaavio oli tuotettu käyttäjävaatimusdokumentin tueksi esisuunnitteluvaiheessa, jonka yhteydessä oli myös tunnistettu tarve käsitellä asiaa jonkinlaisella integraatiokerroksella, koska integroitavien järjestelmien tekniset kyvykkyydet olivat hyvin erilaiset (IT/OT rajapinta). Tämän lisäksi nähtiin etuna, että tulevaisuudessa integroitavia järjestelmiä voi tulla lisää ja niiden tulevia teknisiä reunaehtoja ei voida varmuudella tietää, jolloin skaalautuvuudesta voisi olla hyötyä.

8.2 Arkkitehtuurimalli

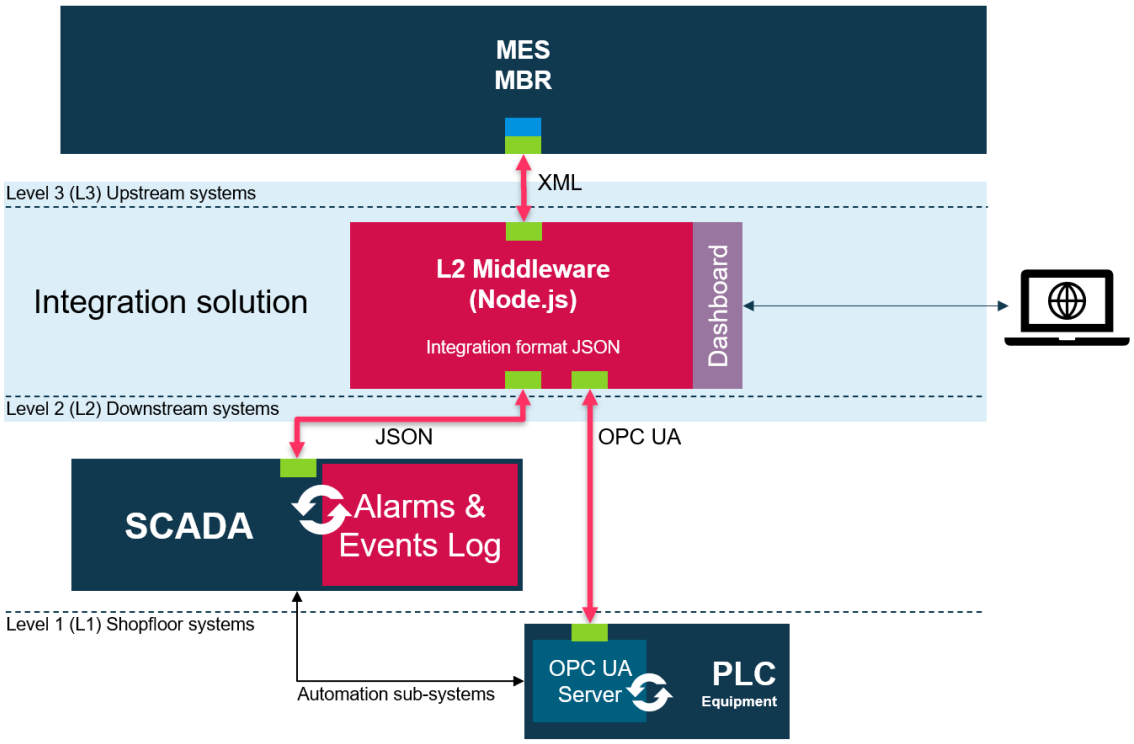
Edellä mainituilla käyttäjävaatimuksilla osoitettiin tarve automaattiselle tiedonvaihdolle tuotannonohjausjärjestelmän (MES) ja tuotantoa ohjaavien ja valvovien järjestelmien (SCADA, PLC) välille sanomapohjaisesti. Teknistä vaatimusta ohjaa Körber Pharman kehittämä ja olemassa olevan Werum PAS-X MES järjestelmän tekninen kyvykkyys (Körber Pharma, n.d.), sekä tilaajan tuotantojärjestelmien ja -koneiden valtava kirjo,

sekä iältään että teknologialtaan. Näiden muuttaminen suoraan tukemaan Körberin laatimaa rajapintaa vasten olisi valtava kustannus, eikä välttämättä isommassa arkkitehtuurikuvassa tarkasteltuna edes kovin kannattavaa, koska rajapinnan muutostilanteissa koneen varsinainen ohjausjärjestelmä olisi avattava aina muutoshallinnan alaiseksi. Tästä syystä esisuunnitteluvaiheessa oli päädytty jonkinlaiseen väliohjelmistoratkaisuun, jolla voidaan MES ja varsinaiset tuotantokoneet erottaa toisistaan ja mahdollistaa toisistaan riippumattomia toiminnallisuuksia.

8.3 Integraatioratkaisu

Väliohjelmistoratkaisun tuli kyetä kommunikoimaan sanomapohjaisesti MES järjestelmän kanssa ja vastaavasti vaihtamaan tietoa esimerkiksi erilaisten automaatiojärjestelmien tiedonsiirtoprotokollien kanssa, kuten esimerkiksi OPC UA (Kuva 17). Esisuunnitteluvaiheessa oli päädytty räätälöityyn väliohjelmistoratkaisuun Node.js pohjaisella ohjelmistoalustalla toteutettavaksi.

Hierarkiassa ylhäällä on Werum PAS-X MES, joka käyttää Körberin omaa REST-arkkitehtuuriin perustuvaa XML-kuvauskielellä toteutettuun MSI (message-base shopfloor integration) sanomarakennetta. Vastaavasti hierarkiassa alhaalla on Siemens WinCC Professional V16 valvomojärjestelmä (SCADA) ja tähän liitetyssä konekannassa on jokaisella oma OPC UA palvelin osana PLC:n ohjelmistoa. Valvomojärjestelmä kattaa yhden tuotanto-osaston toiminnot, jossa on 2 identtistä linjaa ja kummassakin linjassa 4 eri konetta, jonka kautta varsinainen tuotantoprosessi virtaa. Siemens valvomojärjestelmään oli ohjelmoitavissa verkkopalvelun asiakasrajapinta, jolla voitiin toteuttaa REST-arkkitehtuurin mukaisia tapahtumia. Seuraavasta kaaviokuvasta (Kuva 19) on nähtävissä syntyneen järjestelmäintegraation keskeiset tietojärjestelmät, yhteydet ja käytettävien tieto- ja sanomarakenteiden formaatit.



Kuva 19. *Kaaviokuva validoitavan järjestelmäintegraation rajapinnoista ja sanomakielistä.*

Werum PAS-X MES sanomat käyttävät XML-pohjaista sanomarakennetta, joka muunnetaan Node.js integraatioratkaisussa JSON-formaattiin, jotta sen kulku on hallittavissa Node.js sovelluksen sisällä objektimuodossa. Siemens SCADA järjestelmän rajapinta ohjelmoitiin tukemaan suoraan JSON-formaattia, jolloin integraatioille tyypillisiä formaattimuuntimia ei tässä rajapinnassa tarvittu. Sanomaliikenteen valvontaa ja seuraamista varten Node.js alustalle luotiin web-sivu (Dashboard), jonka avulla voidaan esittää sanomaliikenteeseen liittyviä tilastoja ja lokitietoja avustamaan esimerkiksi virhetilanteiden selvityksissä.

8.4 Tilaajan integraatioihin liittyvät motivaatiotekijät

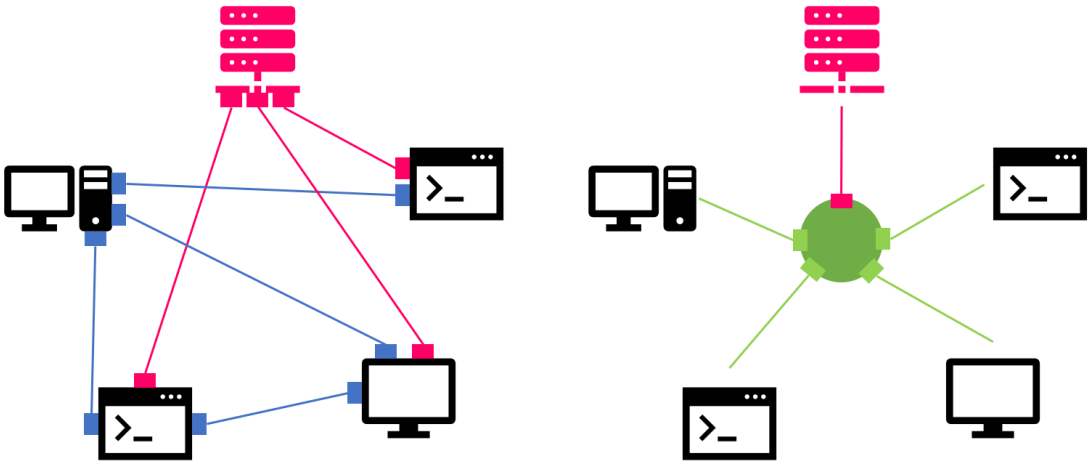
GMP vaatii, että lääkevalmisteen elinkaarta, ts. laatua tukevaa tietoa tulee kerätä ja tallentaa eräpöytäkirjalle, jolla osoitetaan tuotannon virheettömyys (EudraLex Volume 4. n.d.). Tuotannon digitalisoituessa tallennusta tehdään automaattisesti, mutta myös manuaalisesti tuotanto-organisaation eri työntekijöiden toimesta. Tilaajan MES (Manufacturing Execution System) ja ERP (Enterprise Resource Planning) tarjoavat toiminnallisuudet eräpöytäkirjojen laadintaan ja tallennukseen. Eräpöytäkirjalle tuotantoa suorittava ihmiset kirjaavat laadittujen ohjeiden perusteella erilaisia havaintoja ja suoritteita. Näiden lisäksi kriittiset suoritteet allekirjoitetaan kahden eri operaattorin toimesta, ns. neljän silmän periaatteella yrityksen toimintaohjeiden mukaisesti.

Tuotannon digitalisoinnin tavoitteena on vähentää manuaalisten tietojen syöttäminen ja näin alentaa riskejä mm. tietojen eheyteen liittyen. Tämä asetelma monesti vaatii, että eri tasoilla toimivat järjestelmät vaihtavat tietoja keskenään. Erityisesti säädellyssä liiketoiminnassa integrointeihin motivoiva viitekehys on sääntelyn asettamat vaatimukset tietojen eheyteen ja oikea-aikaisuuteen liittyen. (Gambier Sara 2023.)

8.5 Integraatioratkaisuihin liittyviä pohdintoja validointinäkökulmista

Validointityö pohjimmiltaan perustuu toteutettujen ratkaisujen arvioimiseen testaamalla (ks. 5.5 Validointistrategia). Kattavan testaamisen edellytys on, että testauskokonaisuuksia voidaan jotenkin rajata tai paloitella pienempiin osiin, jolloin hallinta ja jäljitettävyys yksittäisten vaatimusten ja testitapausten välillä on kiistatonta (5.7 Validointiprosessi). Erillinen järjestelmien integraatiokerros väliohjelmiston avulla mahdollistaa eri osapuolien testaamisen ilman toista osapuolta, kun hyödynnetään joko väliohjelmiston omia siihen mahdollisesti ohjelmoituja simulointiominaisuuksia tai siihen soveltuvia työkaluohjelmistoja, tai niiden yhdistelmiä. Erillinen integraatiokerros antaa mahdollisuuden myös tutkia sanomaliikennettä sisältäpäin, eli lasilaatikkotestaamisen periaatteilla. Erityisesti tilanteissa missä kohde tai lähdejärjestelmien ”sisälle” ei ole mahdollista nähdä, on kyseinen ominaisuus hyödyllinen etu. Tällainen tilanne voi syntyä silloin kun esimerkiksi kolmannen osapuolen toimittamia valmiita tuotantolaitteita integroidaan.

Validoinnin kannalta on olennaista, että muutoshallintaa suunniteltaessa muutoksen vaikutusalue voidaan rajata tarkasti, jotta olennaiset potilasturvallisuuteen vaikuttavat riskit voidaan tunnistaa luotettavasti (ks. 5.4 Laaturiskien tunnistaminen). Erillinen integraatiokerros erottaa kohde- ja lähdejärjestelmät toisistaan, jolloin vaikutusalue on tarkkarajaisempi. Asian painoarvo korostuu, jos järjestelmäintegraatiot on tehty hajautetusti pisteestä pisteeseen tyyppisesti ns. spagetti-integraatiolla, ks. kuva alla (Kuva 20), jolloin muutoksen arvioitava ja testattava vaikutusalue voi olla ilmeistä laajempi. Ongelma on keskeinen, kun testaukseen sovelletaan automatiikkaa, osaan rajapinnoista ei automatiikka välttämättä yllä tai ei ole siihen soveltuvaa teknologiaa ollenkaan, jolloin käsin tehtävän testauksen osuus voi olla merkittävä hidastava tekijä.



Kuva 20. *Yksittäisen järjestelmän muutosvaikutus (punaisella) spagetti-integraation (vasemmalla) ja keskitetyn integraation välillä (oikealla).*

Edellä mainittujen lisäksi lähde- ja kohdejärjestelmien toimitusprojektit voivat sijaita hyvinkin eri aikajanoilla, jolloin hajautettujen (pisteestä pisteeseen) ratkaisujen yhteensopivuutta toisiinsa voidaan testata ja kehittää vasta kun kaikki järjestelmät ovat käytössä, ellei saatavilla ole käyttöön soveltuvia simulaatiotyökaluja. Simulaatiotyökalut eivät kuitenkaan täysin poista tarvetta kokonaistoiminnan testaamiselta, mutta helpottavat testien suorittamista laajemmalla aikahorisontilla vähentämällä järjestelmien välisiä riippuvuuksia kehitysvaiheessa. Keskitetyssä järjestelyssä integraatiomuuntimia tarvitaan vain yksi jokaista järjestelmää kohden ja sen perustoiminnallisuudet voidaan testata ilman toista osapuolta, koska hyvässä keskitetyssä integraatiossa kaikki tietovirrat muunnetaan yhteiseen väliformaattiin (Tähtinen, 84–87). Keskitetty ratkaisu vain kahden järjestelmän integroimiseksi voi toisaalta olla liian raskas, mutta edellyttää näiltä kahdelta järjestelmältä täyttä yhteensopivuutta ja muunneltavuutta liiketoimintarpeiden muuttuessa.

9. DEVOPS INTEGRAATIORATKAISUN ELINKAAREN HALLINNAN VÄLINEENÄ

Ketterässä ohjelmistokehityksessä on luontaista se, että projektin alussa ei tiedetä minkälaisia lopullisen tuotteen ominaisuudet tulevat olemaan tai tulisi olla. Alkuun pääsemiseksi täytyy määritellä pienimmän toimivan kokonaisuuden ominaisuudet (minimum viable product, MVP), jotta voidaan nopeasti oppia mikä toimii ja mikä ei (Ries 2011). Tämän työn kannalta pienimmän toimivan kokonaisuuden määritelmää edustaa tilaajan laatima URS (ks. 8.1 Käyttäjävaatimukset). Tilaajan projektin ja toimintaympäristön osittaisen tutkimuksellisen ja kehityksellisen luonteen vuoksi integraation lopullisia käyttäjävaatimuksia ei voida määrittää tarkasti ilman käyttökokemuksien kerryttämistä. Tätä varten on luontevaa luoda järjestely elinkaaren hallitsemiseksi ketterästi.

9.1 DevOps rajoitteet

DevOps yhdistää sekä ketterän ohjelmistokehityksen mahdollistavat käytännöt, sekä IT operaatioihin liittyvät toimenpiteet CI/CD työnkulkujen avulla automatisoimalla (ks. 3.2 DevOps). Lääketeollisuus on myös tunnustanut ketterät menetelmät osaksi säädellyn liiketoiminnan prosesseja ja näitä tulkitsevia julkaisuja on enenevissä määrin saatavilla (ks. 3 Ketterä ohjelmistokehitysmenetelmä). GMP:n kannalta kaikkia automatisoitavissa olevia prosesseja ei kuitenkaan voida täysin automatisoida. GMP vaatii laadunvalvonnalle täydellisen itsenäisyyden, jota pidetään keskeisenä periaatteena laadun tyydyttävän tason saavuttamiseksi (EudraLex Volume 4 Good Manufacturing Practice Chapter 6 2014, 3). Näin ollen GMP-ympäristöissä ohjelmiston laatuun liittyviä kontrolleja ei voida täysin automatisoida CI/CD työkuluilla. Käytännössä laadunhallintaprosessissa pitää olla automatiikasta riippumattomia hallinnollisia toimenpiteitä, jotka käytännössä tarkoittavat ihmisen tekemiä tarkastuksia tai testauksia validointi- ja laaduntarkastusvaiheissa.

9.2 Testitapaukset ja testausautomaatio

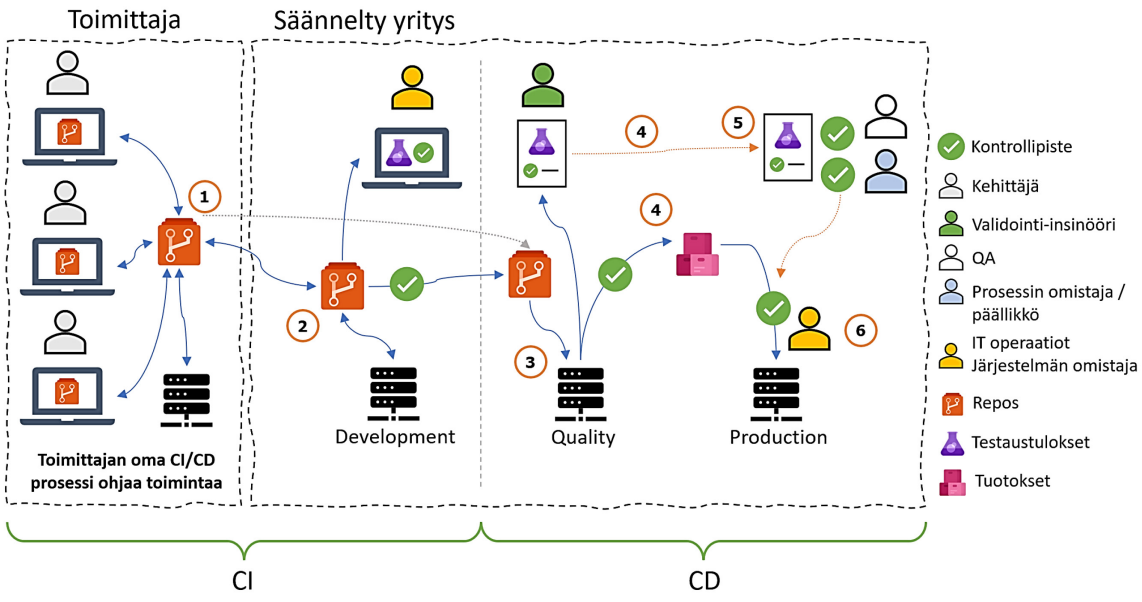
Validointityön keskeinen osa on riskiarvion perusteella muodostuneet testitapaukset ja niiden testaaminen, jolla varmistetaan, että tunnistetut riskit ovat siedettävällä tasolla. Testitapaukset laaditaan siten, että tulos on tulkittavissa tosi/epätosi (Pass/Fail) tyyppisesti vertailemalla testin tuloksia odotusarvoon. Mikäli tulos ei vastaa odotusarvoa, testi

saa tulokseksi epäonnistunut tai epätosi. Joissakin tilanteissa tuloksen tukena on todisteita, kuten kuvakaappauksia tai tulosteita, mutta erityisesti virhetilanteissa niiden kerääminen on keskeistä. (ISPE 2nd, 217.)

GMP:n kannalta varsinaisiin käyttäjähyväksyntätesteihin (UAT) käytetyt testausautomaatiolla suoritettavat testitapaukset tulee arvioida validointi-insinöörin toimesta ensimmäisellä käyttökerralla ja muutoksien yhteydessä tarkastamalla tulokset manuaalisesti tai vaihtoehtoisesti testauskoodit ennakkoon arvioimalla, ei siis riitä, että testausohjelmiston kehittäjä arvioi ne (EudraLex Volume 4 Good Manufacturing Practice: Annex 11 2011, 2). Arviointien jälkeen testaustyökalujen tuottamat tulokset ovat samanarvoisia validointityössä tarvittavien muiden testituloksien kanssa, joten ne ovat keskenään vertailukelpoisia. Testausautomaatioon käytettävät testitapaukset ovat ohjelmakoodia, joten niiden lähdekoodi pitää versioida jäljitystä varten (ks. kappale 6.1.1 Testitapausten jäljitettävyys vaatimuksiin). Näillä toimenpiteillä mahdollistetaan testausautomaatiotapauksien uudelleenkäyttö esimerkiksi regressiotestaamisessa ohjelmistoon tehtävien muutoksien yhteydessä.

9.3 Soveltuva CI/CD-prosessi

GMP:n kannalta on siis olennaista, että CI/CD prosessiin on liitetty manuaalisia kontrollipisteitä, jolla voidaan osoittaa laadunvarmistustoimenpiteiden itsenäisyys (ks. kappale 9.1 DevOps). Seuraavassa kuvassa (Kuva 21) on esitetty mahdollinen GMP:tä tukeva CI/CD prosessi.



Kuva 21. *CI/CD prosessi, jossa GMP vaatimuksia täydentäviä kontrollipisteitä.*

1. Toimittajan kehitystiimi työskentelee ohjelmistokehityksen parissa toimittavan yrityksen omien sisäisten hallinnollisten prosessien mukaan ja tallentaa ohjelmakoodin ja yksikkötestitapaukset yrityksen paikalliseen versiohallintaan.
2. Ohjelmiston julkaisukelpoinen versio ladataan tilaajan kehitysversiohallintaan ja tilaajan edustaja testaa toimivuuden tilaajan omalla kehityspalvelimella. Vaihetta ei voida suorittaa, mikäli tilaajalla ei ole erillistä kehitysympäristöä.
3. CI-putken toimittama ohjelmisto ladataan pääversiohallintaan, joka käynnistää CDE-vaiheen ja toimittaa ohjelmiston laatujärjestelmään ja suorittaa validointisuunnitelman mukaiset automaattiset käyttäjähyväksyntätestitapaukset (UAT).
4. Validointi-insinööri suorittaa manuaaliset käyttäjähyväksyntätestitapaukset (UAT) ja laatii testausraportin, jonka jälkeen sallii testivaiheen läpäisseen tuotoksen tallentamisen tuotosversiohallintaan julkaisua varten tai palauttaa toimitetun tuotoksen takaisin valmisteluun.
5. Prosessin päällikkö ja laatutoiminnot tarkastavat työn virheettömyyden ja antavat luvan julkaista syntyneen tuotoksen tai pysäyttävät prosessin, mikäli toimituksessa on puutteita.
6. IT operaatiot tarkastavat, että julkaisu on onnistunut ja järjestelmä toimii normaalisti (tämä vaihe voitaisiin myös automatisoida CD-vaiheen päätteeksi).

Edellä kuvatussa työnkulussa CI-vaihetta edustaa kohdat yksi ja kaksi, CDE-vaihetta kolme ja neljä, sekä CD vaihetta kohdat viisi ja kuusi. Riippuen ratkaisuista ja sovituista käytännöistä CI-prosessin vaihetta kaksi ei välttämättä voida suorittaa, jolloin kehitystiimi toimittaa ratkaisun suoraan laatujärjestelmään testattavaksi. Tässä on kuitenkin riskinä se, että ilmeisiä virheitä ja niistä syntyviä poikkeamia tulee tarpeettomasti, jotka olisivat mahdollisesti havaittu jo vaiheessa kaksi. Esitetyssä CI/CD työkulussa on GMP:n vaatimia kontrollipisteitä CI putkessa sekä CD putkessa, siten että, GMP:n vaatimusten mukaisesti osoitus laadunhallinnan itsenäisyydestä täyttyy. Tämä lisäksi käyttäjäorganisaatiolle on luotu, samoja mekanismeja käyttäen, edellytykset liiketoimintariskien hallinnalle.

Kontrollipisteiden osalta työssä “Towards RegOps: A DevOps Pipeline for Medical Device Software” (Toivakka, Granlund, Poranen, Zhang 2021, 10–11) on tultu samankaltaisiin johtopäätöksiin, siitäkin huolimatta, että aihe käsittelee lääkinnällisten laitteiden sovelluskehitystä. Edellä mainitun lisäksi työssä mainitaan, että se on vain osa kokonaisratkaisua, eikä täysin täytä lääkinnällisten laitteiden kehitystyön regulatiivisia vaatimuksia. Työssä ei esimerkiksi esitetä ketterän kehitysvaiheen toimenpiteitä, eli vaatimusmäärittelyjä, eikä niihin liittyviä riskienhallintaan liittyviä toimenpiteitä, jotka ovat yhtä lailla osa DevOps käytäntöjä, mutta ennen kaikkea se on regulatiivinen vaatimus.

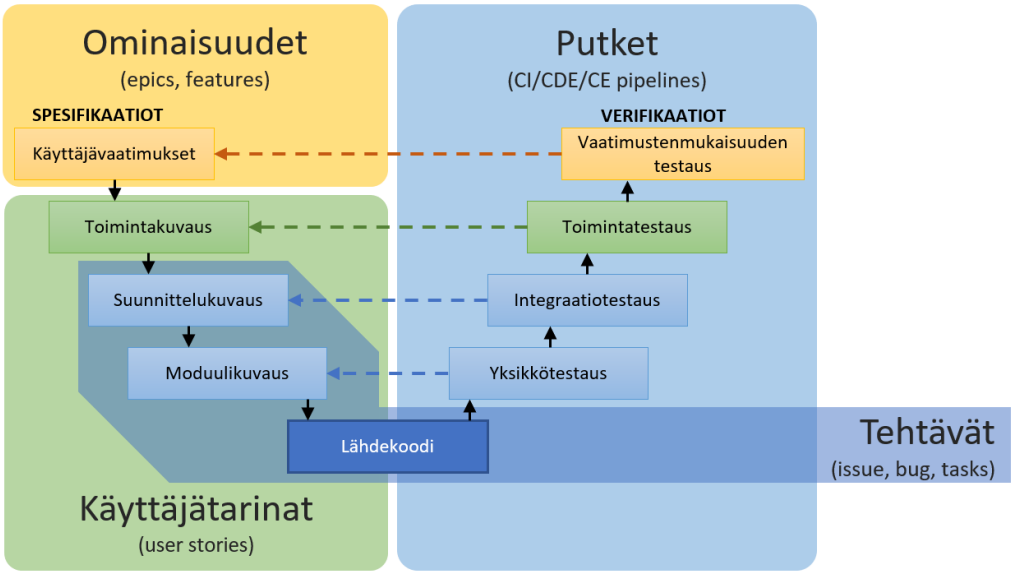
10. ELINKAAREN HALLINNAN JA TESTAUKSEN TYÖKALUT

Tässä kappaleessa työ etenee toteutusvaiheeseen eli ehdotukseen GMP mukaisen elinkaaren hallintaan liittyvien prosessien hallitsemiseksi ketterien menetelmien ja DevOps CI/CD työnkulkujen avulla, jota myöhemmin kutsutaan yleisesti DevOps malliksi. Käytännöt ja CI/CD putket toteutettiin tilaajan Azure DevOps ympäristöllä, mutta sinänsä elinkaarimalli voidaan toteuttaa millä tahansa DevOps työkalulla, josta löytyvät tarvittavat ominaisuudet.

Kappaleessa 5 esiteltiin, että perinteisesti lääketeollisuuden tietojärjestelmäprojekti alkaa aikeesta tietokoneistaa jokin liiketoimintaprosessi ja yleensä tämä vaihe sisältää jonkin esisuunnitteluvaiheen, jossa tarpeen laajuutta, budjettia ja mahdollisia toimittajia arvioidaan. Tämän jälkeen tarpeesta laaditaan yksityiskohtainen käyttäjävaatimusspesifikaatio (URS). URS hyväksynnän jälkeen laaditaan riskiarvio ja tämän perusteella validointistrategia, joka tarkoittaa mm. testausstrategiaa. Validointistrategialla, jota toteutetaan spesifikaatioverifikaatio-mallilla, pyritään osoittamaan, että toteutettu tietokoneistettu järjestelmä potilasriskien kannalta soveltuu aiottuun käyttöön ja täyttää tietoturvan, tiedon eheyden ja oikea-aikaisuuden viranomaisvaatimukset.

Kappaleessa 3 esitetyssä ketterässä elinkaarimallissa ei voida toimia lääketeollisuuden perinteisillä tavoilla, koska kaikkia tulevia ominaisuuksia ei vielä tarkalleen tiedetä (ks. 9 DevOps Integraatioratkaisun elinkaaren hallinnan välineenä), ts. URS ei hankkeen alussa välttämättä sisällä kaikkia tarvittavia vaatimuksia. DevOps elinkaarimallissa ensimmäisen iteraatiokierroksen URS vastaa pienimmän toimivan kokonaisuuden toteutusta, eli MVP:tä. Kokemuksien karttuessa laadittuja käyttäjävaatimuksia toteutetaan lisää ja URS päivittyy jokaisen tuotantoon viedyn iteraatiokierroksen jälkeen, jonka avulla ylläpidetään GMP:n mukaista dokumentaatiota (ks. Kuva 23).

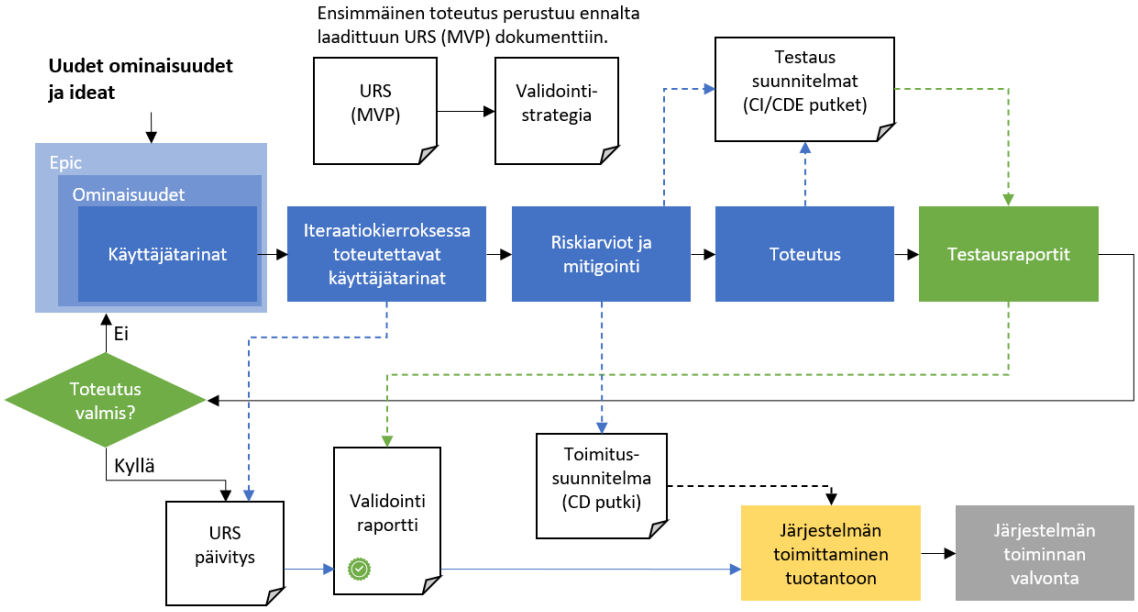
Edellä mainitun haasteen lisäksi, DevOps mallin terminologia eroaa verifikaatiospesifikaationmallin (v-malli) terminologiasta. Seuraavassa kuvassa (Kuva 22) on esitetty v-mallin keskeiset termit ja tietohierarkiat, sekä niitä vastaavat termit ja tietohierarkiat DevOps elinkaarimallissa.



Kuva 22. *Verifikaatiospesifikaatiomallin ja DevOps-mallin vastaavuudet.*

Esitetyssä DevOps-mallissa, ominaisuudet vastaavat perinteisen mallin käyttäjävaatimuksia. Käyttäjätarinat vastaavasti perinteisen mallin spesifikaatioita, vastaavasti verifikaatioita vastaa erilaiset putkien sisältämät automaattiset ja manuaaliset testitapaukset. Edellä mainittujen lisäksi ominaisuuksia voidaan hallita Epic-tason avulla, ikään kuin aihealueen runkona. Epic vastaa perinteisen mallin URS dokumentin kappaleita, kuten esimerkiksi kappaletta koskien vaatimuksia tietyltä alueelta, kuten autentikointi. Vastaavasti toteutettava ominaisuus saattaa sisältää useita käyttäjätarinoita ja kukin käyttäjätarina vastaavasti yhden tai useamman tehtävän. DevOps-mallissa lähdekoodia on varsinaisen toteutuksen osalta myös testausautomaation ja putkien toteutuksien osalta sillä erolla, että testikoodien ja putkien koodia varten ei ole varsinaista spesifikaatiota, koska niiden toteutus perustuu riskiarvioihin.

Siinä missä DevOps-mallin terminologia eroaa perinteisestä mallista niin toimintaprosessin on myös erottava perinteisestä spesifikaatioverifikaatiomallista, koska toteutuksen ominaisuuksia jalostetaan ja kehitetään projektin edetessä. Seuraavassa kuvassa (Kuva 23) on ehdotus mahdolliselle GMP:tä tukevalle DevOps työnkululle.



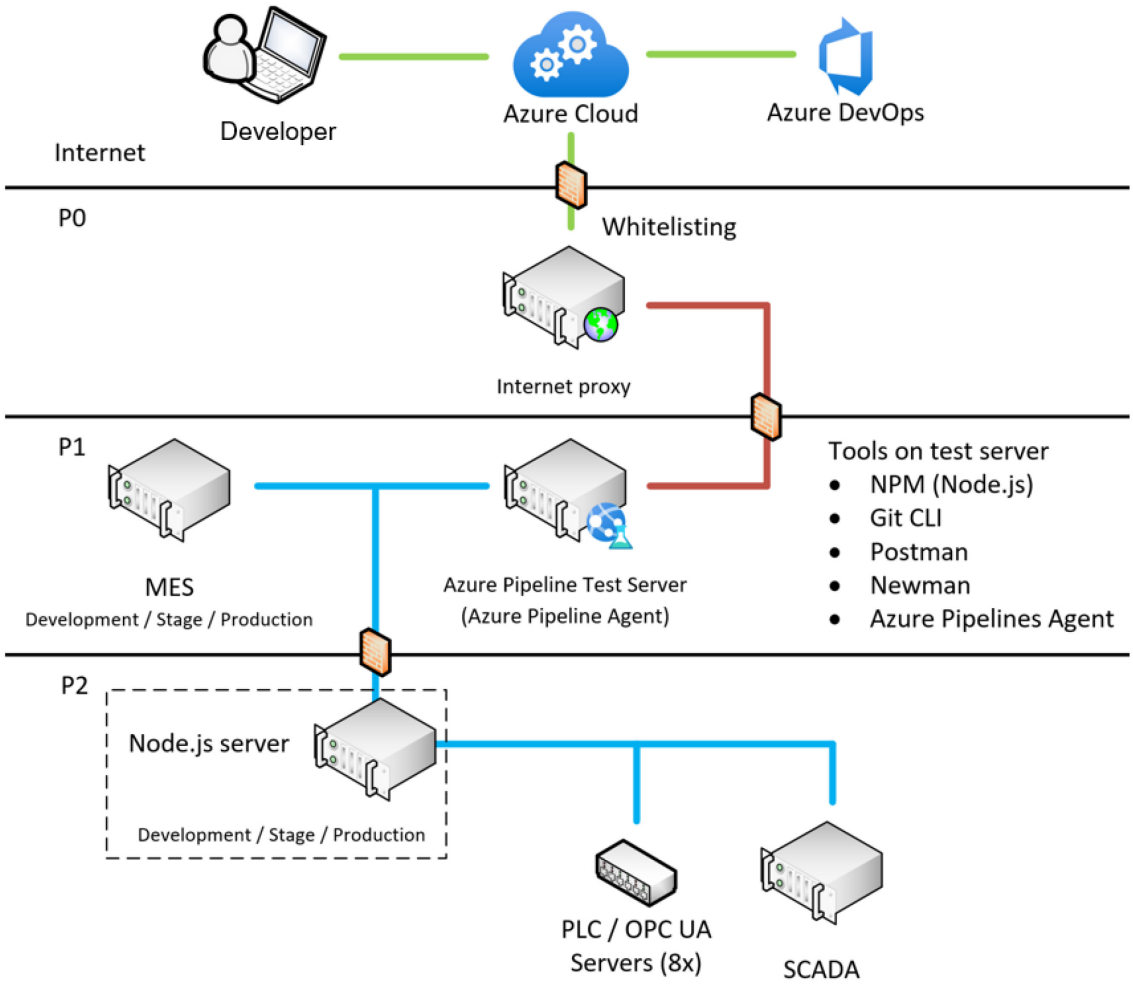
Kuva 23. *Ehdotus DevOps työnkululle tietokoneistettujen GxP-järjestelmien kehittämiseksi ja toimittamiseksi tuotantoon.*

Ensimmäisellä toteutuskerralla käyttäjäorganisaatio laatii URS-dokumentin perinteisiä työnkulkuja noudattaen, joka tässä tapauksessa edustaa MVP-toteutusta, ja jonka perusteella määritetään validointistrategia. Hyväksytyn URS-dokumentin vaatimukset kirjataan käyttäjätarinamuotoon. Tämän jälkeen toteuttava tiimi valitsee ensimmäiseen iteraatiokierrokseen toteutettavat ominaisuudet ja suorittavat niihin liittyvät tehtävät mukaan lukien riskiarviot, mitikointikeinot, sekä testitapaukset. Iteraatiokierroksia suoritetaan tarvittava määrä, kunnes toteutus on niiltä osin valmis. Valmiista toteutuksesta laaditaan validointiraportti ja URS-dokumentti päivitetään. Tämän jälkeen toteutus voidaan viedä tuotantoon toimitussuunnitelman mukaisesti ja lopuksi järjestelmä siirtyy IT-operaatioiden vastuulle. Ensimmäisen toteutuksen jälkeen uudet ominaisuudet laaditaan suoraan käyttäjätarinamuotoon ja iteraatiokierroksia toteutetaan aiemman selosteen mukaan tarvittava määrä, kunnes toteutus viedään jälleen tuotantoon URS-päivityksen ja validointiraportin hyväksynnän jälkeen.

10.1 Automaattisen testauksen järjestelyt

Kappaleessa 8 (Validoitava Järjestelmäintegraatio olennaisilta osin) esiteltyä integraatioratkaisua ei ole mahdollista testata testausautomaation avulla pilvipalvelusta löytyvällä virtuaalikoneella, koska testaamiseen osallistuvat kehitys- laatu- ja tuotantojärjestelmät ovat asennettuna tilaajan suojatussa tuotantoverkossa, johon ei luonnollisesti ole avointa pääsyä. Tällaisessa tilanteessa testauksia suorittava palvelin samassa verkossa on soveltuva ratkaisu. Tätä varten, tuotantoverkkoon asennettiin Pipeline agentille palvelin ja avattiin palomuurista tarvittavat sisäiset yhteydet testauskohteisiin, sekä

pilvessä olevalle Azure DevOps ympäristölle tuotantoverkossa olevan proxy palvelimen kautta. Kuvaus testaus- ja kehitysympäristöstä esitetty seuraavassa kaaviokuvassa (Kuva 24), jossa esitetään ympäristöön kuuluvat MES, Node.js ja SCADA palvelut, sekä niiden väliset verkkoyhteydet.



Kuva 24. *Kaavio testausympäristön keskeisistä osista ja niiden loogisista yhteyksistä toisiinsa nähden, jossa P0 - P2 edustavat tilaajan tuotantoverkon eri jakoalueita (Mukailtu tilaajan sisäisestä dokumentaatiosta).*

Tarvittavan internet liikenteen varmistamiseksi Pipeline agentille avattiin internet proxy palvelimelle reitit ns. valkolistaamalla (whitelisting) tarvittavat internetosoitteet. Näihin tarvittavat säännöt ovat saatavilla Microsoftin Learn dokumentaatiosta (Self-hosted Windows agents 2023).

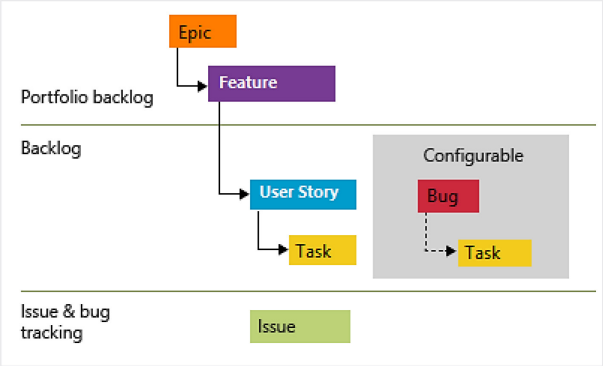
10.2 Työkalut

Integraatioratkaisun toteuttamiseksi tarvitaan koko joukko erilaisia työkaluohjelmistoja tukemaan sovelluksen elinkaarta. GMP:n mukaan työkaluohjelmistoja ei tarvitse validointia, mutta näiden käyttämiseksi ja ylläpitämiseksi vaaditaan hyviä IT käytäntöjä,

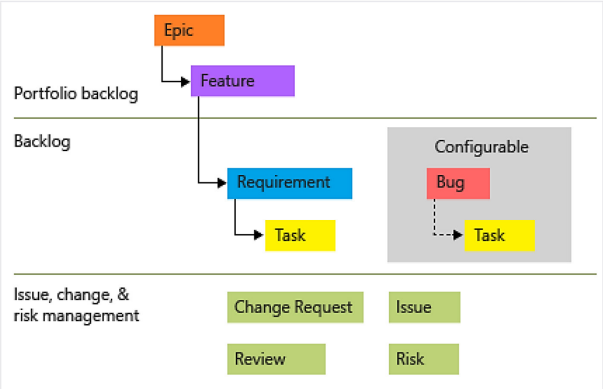
(ks. kappale 6.6 Testausautomaatiovaatimukset). Microsoft Azure DevOps alustan työkalupakkiin kuuluu mm. putket CI ja CD työnkulujen automatisoimiseksi, Git versiohallinta, Kanban taulut, sprintit, sekä näiden lisäksi muita projektihallinnollisia työkaluja (What is Azure DevOps? 2022).

10.2.1 Azure DevOps

Azure DevOps ympäristössä käytettävät ominaisuudet, eli työosiot (work items) riippuvat projektille valitusta tietomallista. Vaihtoehdot ovat Basic, Agile, Scrum ja CMMI. Nämä vaihtoehdot eivät sinänsä vaikuta toteutettaviin CI/CD putkiin, mutta GMP vaatii kehittämiseen käytettävän elinkaarimallin tunnistamista (ks. 5.5 Validointistrategia). Muutoshallinta ja varsinainen riskien tunnistaminen suoritetaan tilaajan validointi- ja muutoshallintajärjestelmissä (ks. kappale 7), tällöin Azure Agile-tietorakennemalli on toteutuksien suunnittelun kannalta riittävä (Kuva 25). CMMI malli (Kuva 26) voisi luoda tarpeetonta päällekkäisyyttä (ks. 5.5 Validointistrategia, 7 Tilaajan validointitestaus käytännössä). Basic tietomalli on liian yksinkertainen ja vastaavasti Scrum ei sovellu tilaajan projektityöskentelymalliin, koska eri vaiheiden ajallinen kesto vaihtelee tuotantolinjojen valmistumisen mukaan.

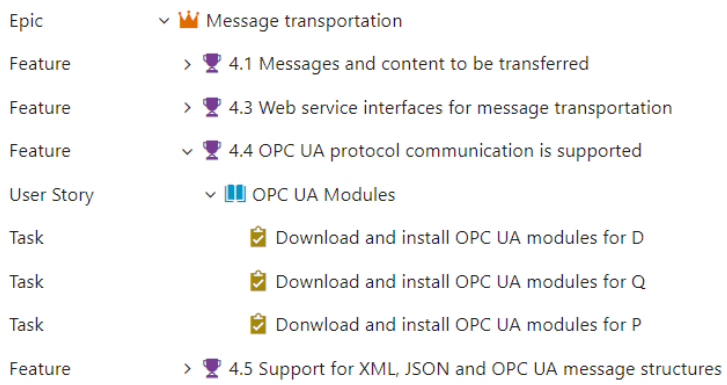


Kuva 25. *Agile tietorakennemalli (Microsoft Learn 2023).*



Kuva 26. *CMMI tietorakennemalli (Microsoft Learn 2023).*

Elinkaaren hallitsemista varten luotiin Azure DevOps työympäristö, johon syötettiin käyttäjäorganisaation laatiman URS dokumentin yksittäiset vaatimukset ryhmittelemällä ne aiheisiin sopiviin suurempiin kokonaisuuksiin eli Epic:hin valitun tietorakenteen mukaisesti (Kuva 25). Varsinaiset URS dokumentin vaatimukset kirjattiin Epic kokonaisuuksien alle ominaisuuksina (Feature), josta pieni ote seuraavassa kuvassa (Kuva 27).



Kuva 27. *Epic, jolla on neljä ominaisuutta (Feature)*

Jokaiselle ominaisuudelle laaditaan vähintään yksi käyttäjätarina (User Story), jossa kerrotaan, kuinka vaadittava ominaisuus toteutetaan ja mitkä ovat hyväksymiskriteerit. Kun käyttäjätarina on laadittu ja hyväksytty, tarinan toteuttamiseksi laaditaan tehtäviä (Task), kuten edellisessä kuvassa (Kuva 27), jolloin varsinainen kehitystyö vaatimuksen kohdalla voi alkaa. Näistä kaikista yhdessä muodostuu toteutettavan järjestelmän spesifikaatiot.

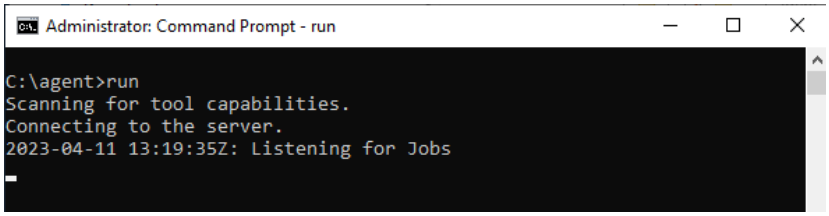
10.2.2 Azure Pipelines

Azure Pipelines toiminnallisuudella suoritetaan jatkuvan integroinnin, jatkuvan toimituksen ja jatkuvan testauksen prosesseja (CI/CD) mm. koodin kääntämiseksi, testaamiseksi ja toimittamiseksi kohdejärjestelmään. Edellytyksenä on, että lähdekoodi on versionhallintajärjestelmässä, kuten GitHub, GitLab tai Azuren mukana tulevassa Repo:ssa. Erilaisia tehtäviä ja testejä voidaan suorittaa käyttämällä monia testauskehyksiä ja palveluita, joka sisältää myös Shell-komentosarjat. (What is Azure Pipelines? 2023)

Azure putket laaditaan YAML kielellä tai vaihtoehtoisesti ne voidaan generoida käyttämällä "classic editoria", joka generoi YAML formaatin mukaisia konfiguraatiotiedostoja (YAML pipeline editor 2023). Konfiguraatiotiedostot tallennetaan Azure:n Repoon, jolloin myös putkien GMP:n vaatimat versiohallinnalliset kontrollivaatimukset täyttyvät.

10.2.3 Azure Pipeline Agent

Azure Pipeline Agent on komentoriviltä ajettava sovellus (Kuva 28), joka suorittaa putkissa määritetyt tehtävät erillisellä testaustietokoneella, esimerkiksi ajamalla erilaisia tehtäviä komentoriviltä. Agentin voi asentaa jatkuvasti käynnissä olevana palveluna, kirjautumisen yhteydessä automaattisesti käynnistyvänä ohjelmana haluamallaan käyttäjätunnuksella tai erikseen tarvittaessa käynnistettävänä ohjelmana.



Kuva 28.*Komentoriviltä käynnistetty agenttisovellus.*

Agentti tarkastaa ensin työkalun kyvykkyydet, kuten määritetyt käyttöoikeusavaimet ja ottaa tämän jälkeen yhteyden Azure DevOps Pipeline palveluun ja jää odottamaan mahdollisia töitä.

10.2.4 Postman

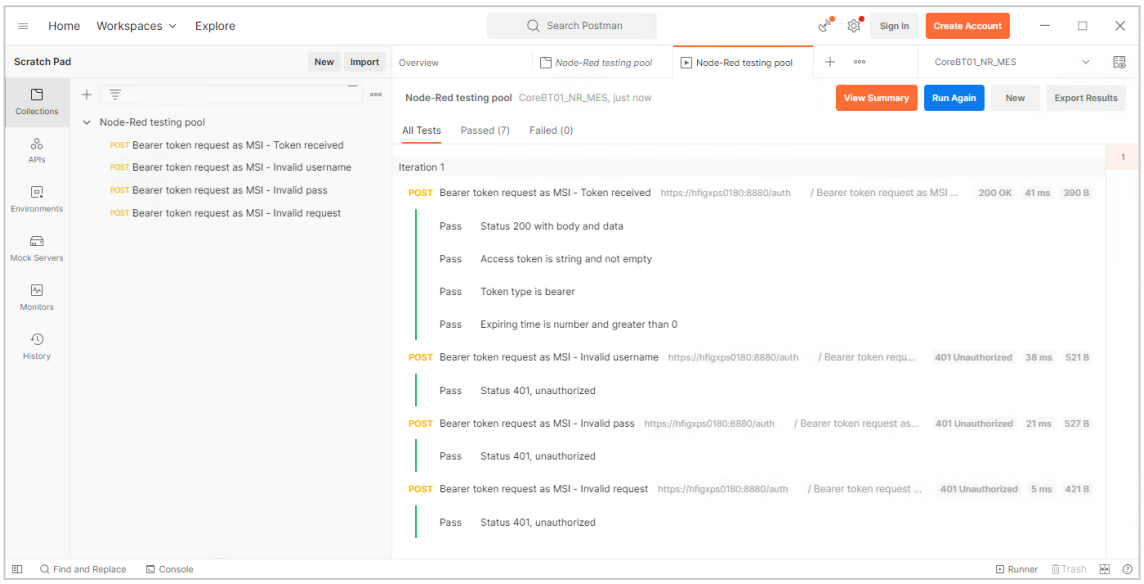
Postman on ohjelmointirajapinta-alusta (API-alusta) sovellusliittymien ja rajapintojen testaamiseen, rakentamiseen ja käyttöön (Postman. n.d.). Postman työkalulla voidaan luoda erilaisia testikokoelmia API-rajapintoihin, jotka jäljittelevät API-asiakasohjelmaa. Suoritetun testitapauksen asiakasrajapinnasta samaa syötettä voidaan validoida JavaScript kielellä kirjoitetuilla testeillä (Writing tests 2023).

```
pm.test("Status 200 with body and data", function() {
    pm.response.to.have.status(200);
    pm.response.to.be.withBody;
});
```

Ohjelma 1. *Postman ohjelmalla kirjoitettu testitapaus, joka tarkastaa API-rajapinnan palauttaman tilan ja sen että vastauksella oli ylipäätään sisältöä.*

Postman työkalulla luodut testitapaukset ovat käytännössä JavaScript koodia (Ohjelma 1) ja tallentuvat Postman ohjelmaan paikallisesti (Kuva 29) ja sen omaan pilvipalveluympäristöön yhteystyön mahdollistamiseksi muiden kehittäjien kanssa.

Postmanilla tehdyt testitapaukset tuottavat mustalaatikkotestauksen kaltaisia tuloksia, eli testattavaan rajapintaa annetaan syötteitä ja tämän jälkeen tutkitaan saatuja tuloksia odotusarvoon vertailemalla tosi/epätosi (Pass/Fail) tyyppisillä päättelyillä, joka tukee GMP:n mukaisia testikäytäntöjä.



Kuva 29. *Postman sovelluksella kehitettyjä ja suoritettuja testitapauksia.*

Postman ei tue versionhallintaa eikä muita elinkaaren hallintaan liittyviä palveluja ilman internet-pilvessä olevia työtiloja. Sen sijaan, testitapaukset voidaan viedä ohjelman ulkopuolelle JSON muodossa ikään kuin lähdekoodina ja tallentaa Azuren versiohallintaan. Elinkaaren hallinnan kannalta testitapauksien pirstaloituminen useampaan eri käyttöympäristöön ei ole GMP-vaatimuksien ja validointityön kannalta kovin ideaali tilanne, koska testitapaukset pitää pystyä jäljittämään aukottomasti käyttäjävaatimuksiin (ks. kappale 5.5 Validointistrategia). Tästä syystä, Postman pilvipalvelun käyttö ei ole optimaalisin ratkaisu.

10.2.5 Newman

Newman on komentorivillä ajettava työkaluohjelma, jonka avulla voi ajaa Postman testikokoelmia suoraan komentoriviltä, joka tukee jatkuvan integroinnin (CI) työnkulkuihin liittyvissä testitapauksissa. Newman ajaa testikokoelmat komentorivillä samalla tavalla kuin ne ajettaisiin Postman ohjelmassa ja testitulokset voidaan tulostaa erilaisille raporttimuodoille (Running collections on the command line with Newman 2023). Newman sovellus tuo ratkaisun siihen, että Postman sovelluksella kehitetyt testitapaukset ovat suoritettavissa Pipeline Agentilla shell-komentoina.

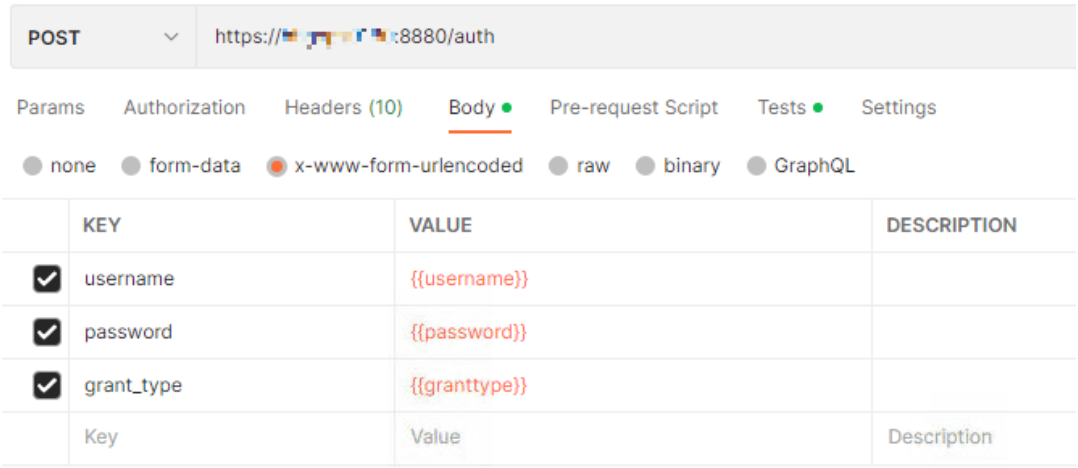
10.3 Testausautomaation soveltaminen

Integraatioon liittyviä testausautomaatiotapauksia voidaan soveltaa väliohjelmiston rajapintojen testaamiseen suorittamalla Postman sovelluksella kehitettyjä testitapauksia. Tämänkaltaiset testit vastaavat suoraan URS dokumentaatiossa esitettyihin vaatimuk-

siin, koska siinä ei yleensä oteta kantaan siihen, miten sovelluksen sisäisen logiikan tulisi toimia. Ohjelmiston sisäinen toiminta, testaaminen ja näihin liittyvien testaustuloksien osoittaminen tilaajalle on yleensä vastuullistettu ohjelmiston valmistajalle, jolloin niitä voidaan myös hyödyntää osana validointityötä, ks. kappale 6.2 Toimittajan tekemä testaus.

10.4 Testitapausten kehittäminen ja suorittaminen paikallisesti

Postman työkalun avulla kehitetään varsinaiset rajapinnat ja niihin liittyvä testitapaukset, johon kuvataan rajapintakysely ja sarja yksittäisiä testitapauksia siitä mitä rajapinnan pitää palauttaa kulloisellakin kyselyllä. Kuvissa alla (Kuva 30, Kuva 31) on Postman kehitystyökalulla luotu kysely ja siihen liittyvät testitapaukset. Postman testitapaukset rinnastuvat GMP:n käsikirjoitettuihin testeihin.



Kuva 30. *Postman kehittimessä oleva POST metodilla luotu kysely, joka pyytää autentikointiavainta.*

Postman tukee ympäristömuuttujia ja niitä voidaan käyttää osana kyselyjä "mustache"-templaatti-järjestelmän mukaisesti, eli tuplakaarisulkeiden avulla. Ympäristömuuttujien käyttö mahdollistaa samojen testien suorittamisen eri osoitteissa oleville palveluille. Eri kehitysympäristöjen ympäristömuuttujat voidaan myös viedä JSON muotoisena versiohallintaan, jolloin CI/CD putkien testausvaiheissa voidaan käyttää samoja testitapauksia eri kehitysympäristöissä vain ympäristömuuttujia vaihtamalla, tämä keventää validointityötä testitapauksien arviointiin liittyen.

POST https://[redacted]:8880/auth

Params Authorization Headers (10) Body Pre-request Script Tests Settings

```
let jdata = pm.response.json();
let URS = "URS001: Bearer token request - ";

pm.test(URS + "Status 200 with body and data", function() {
    pm.response.to.have.status(200);
    pm.response.to.be.withBody;
});

pm.test(URS + "Access token is string and not empty", function () {
    pm.expect(jdata.access_token).to.be.a("string").and.not.be.empty;
});

pm.test(URS + "Bearer token request - Token type is bearer", function () {
    pm.expect(jdata.token_type).to.be.eql("bearer");
});

pm.test(URS + "Bearer token request - Expiring time is number and greater than 0", function () {
    pm.expect(jdata.expires_in).to.be.a("number");
    pm.expect(jdata.expires_in).to.be.greaterThan(10);
});
```

Kuva 31. *Aiemman kuvan (Kuva 30) kyselyyn liittyvät testitapaukset, jotka validoivat rajapinnan palauttaman vastauksen.*

Koodi varsinaisille testitapauksille kirjoitetaan JavaScript:llä käyttämällä Postman tuotteen tarjoamaan pm objektia (Kuva 31). Objektin syntaksi on hyvin kuvaavaa ja koodia on mahdollista oppia ymmärtämään varsin helposti myös henkilöt, jotka eivät ole tottuneet kehittämään koodia, kuten validointi-insinöörit. Kehitettyjä testitapauksia on helppo testata suoraan käyttöliittymästä (Kuva 32), josta näkee myös rajapinnan palauttaman datan sisällön (Kuva 33).

Node-Red testing pool CoreBT01_NR_MES, just now

All Tests Passed (7) Failed (0)

Iteration 1

POST Bearer token request as MSI - Token received https://hfigxps0180:8880/auth / Bearer token request as MSI - Token received

Pass URS001: Bearer token request - Status 200 with body and data

Pass URS001: Bearer token request - Access token is string and not empty

Pass URS001: Bearer token request - Token type is bearer

Pass URS001: Bearer token request - Expiring time is number and greater than 0

POST Bearer token request as MSI - Invalid username https://hfigxps0180:8880/auth / Bearer token request as MSI - Invalid username

Pass URS001: Bearer token request - Status 401, unauthorized

POST Bearer token request as MSI - Invalid pass https://hfigxps0180:8880/auth / Bearer token request as MSI - Invalid pass

Pass URS001: Bearer token request - Status 401, unauthorized

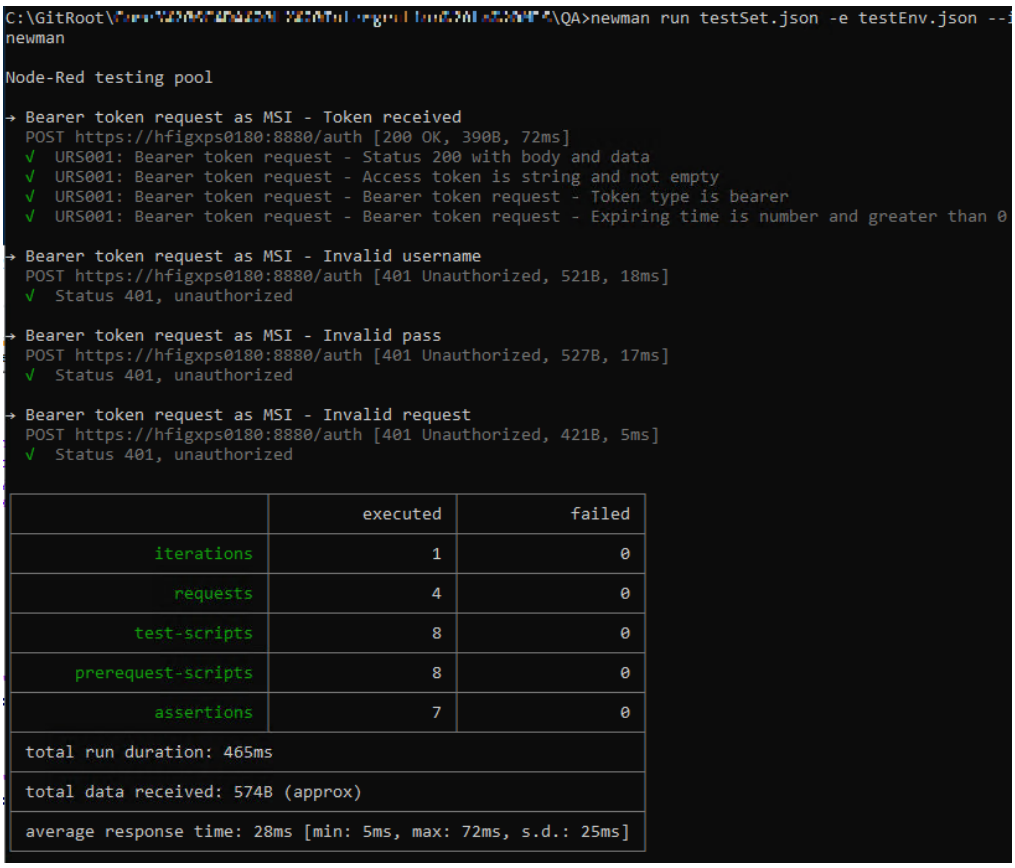
POST Bearer token request as MSI - Invalid request https://hfigxps0180:8880/auth / Bearer token request as MSI - Invalid request

Pass URS001: Bearer token request - Status 401, unauthorized

Kuva 32. *Postman testiajon palauttama raportti testien tuloksista.*

Kuva 33. *Postman kyselyn tuottaman rajapintakyselyn antama palaute, joka tässä tapauksessa on JSON tyyppinen.*

Postman testitapauksien ajaminen suoraan komentoriviltä ei onnistu, jota tarvitaan testien automatisoimiseksi CI/CD putkien avulla. Tätä varten valmiit testitapaukset viedään JSON muodossa versionhallintaan. Tämän lisäksi versiohallintaan on vietävä ympäristömuuttujat, jotta testitapaukset kohdistuvat oikein haluttuihin ympäristöihin. Testitapauksien suorittaminen komentoriviltä Newman ohjelmalla tuottaa vastaavat tulokset kuin Postman käyttöliittymässä (Kuva 34).



Kuva 34. *Newman ohjelman tuottama raportti suoritetuista testeistä.*

Komentoriville tulostunutta raporttia ei voida tallentaa ja versioida, joten tarvitaan erillinen raportti. Tätä varten Azure Pipelines tukee useita eri raporttiformaatteja, kuten JUnit, VSTest, XUnit V2, NUnit ja CTest (Publish test results to Azure Pipelines 2023). Newman sovelluksen raporttitoimintoa hyödyntämällä yhdessä NPM:stä löytyvällä JUnit raportointilaajennuksella "newman-reporter-junitfull" saadaan testitulokset tulostettua XML muotoisina JUnit raportteina (Ohjelma 2).

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites name="Node-Red testing pool" tests="4" time="0.130">
  <testsuite name="Bearer token request as MSI - Token received" id="64544545-400d-4cf8-b6e7-21c07d202e52" timestamp="2023-04-16T08:00:30.257Z" tests="4" failures="0" errors="0" time="0.077">
    <testcase name="URS001: Bearer token request - Status 200 with body and data" time="0.077" classname="NodeRedTesting-Pool"/>
    <testcase name="URS001: Bearer token request - Access token is string and not empty" time="0.077" classname="NodeRedTest-ingPool"/>
    <testcase name="URS001: Bearer token request - Token type is bearer" time="0.077" classname="NodeRedTestingPool"/>
    <testcase name="URS001: Bearer token request - Expiring time is number and greater than 0" time="0.077" class-name="NodeRedTestingPool"/>
  </testsuite>
  <testsuite name="Bearer token request as MSI - Invalid username" id="05d0e289-f74f-4a72-8bc3-1ff7c28db190" timestamp="2023-04-16T08:00:30.257Z" tests="1" failures="0" errors="0" time="0.021">
    <testcase name="URS001: Bearer token request - Status 401, unauthorized" time="0.021" classname="NodeRedTestingPool"/>
  </testsuite>
  <testsuite name="Bearer token request as MSI - Invalid pass" id="3bf5f189-f894-45bb-95ba-fd205e8ea84d" timestamp="2023-04-16T08:00:30.257Z" tests="1" failures="0" errors="0" time="0.027">
    <testcase name="URS001: Bearer token request - Status 401, unauthorized" time="0.027" classname="NodeRedTestingPool"/>
  </testsuite>
  <testsuite name="Bearer token request as MSI - Invalid request" id="cb3b8738-1263-42b3-b338-dd3e0c615ea4" timestamp="2023-04-16T08:00:30.257Z" tests="1" failures="0" errors="0" time="0.005">
    <testcase name="URS001: Bearer token request - Status 401, unauthorized" time="0.005" classname="NodeRedTestingPool"/>
  </testsuite>
</testsuites>
```

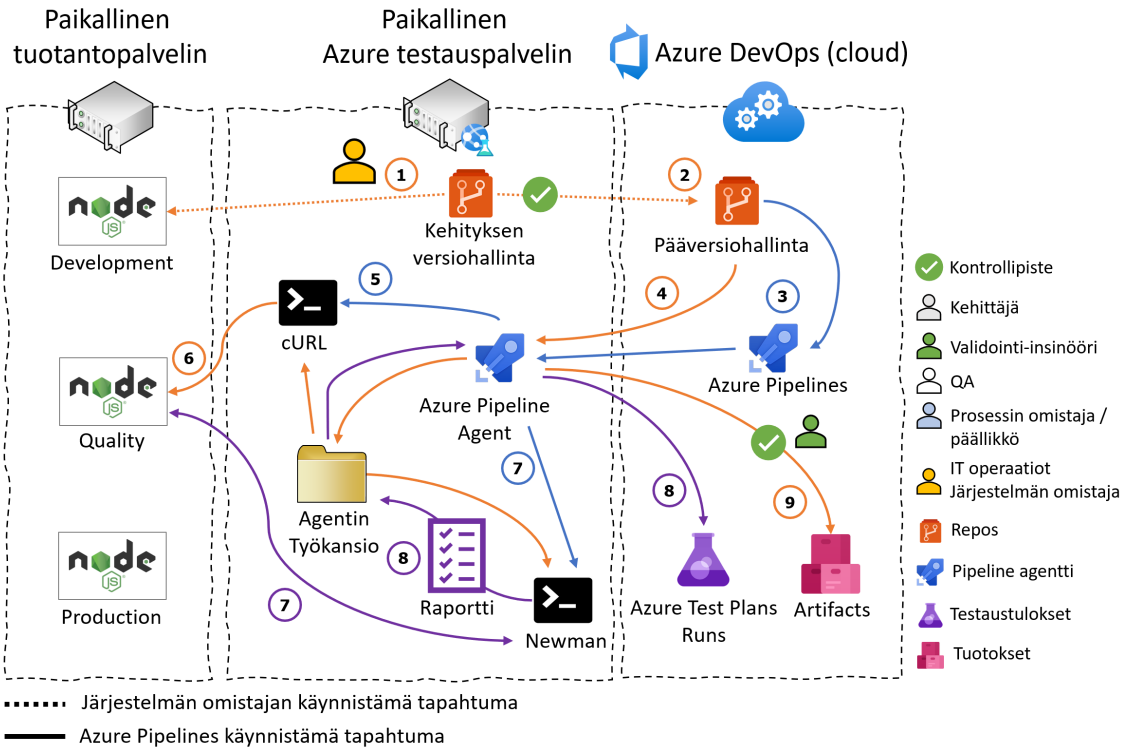
Ohjelma 2. *Raporttilaajennuksen tuottama XML muotoinen JUnit raportti.*

Esitetyllä ratkaisulla Newman sovelluksella voidaan ajaa paikallisesti Postman sovelluksella kehitettyjä JSON muotoisia testitapauksia Azure Pipeline Agentin ohjaamana niin, että siitä saadaan muodostettua Azure DevOps yhteensopivia raportteja.

10.5 CI/CD putkien tekninen toteutus

Tässä kappaleessa selostetaan seikkaperäisesti CI/CD prosessien tekninen toteutus annetuilla työkaluilla, jolla suoritetaan käyttäjähyväksyntätestaus (UAT), joka käytännössä sisältää integraatiorajapintaan liittyviä automaattisia testitapauksia, sekä manuaalisia toimenpiteitä. Selosteista on jätetty kolmannen osapuolen prosessit kuvaamatta, koska ne eivät ole tilaajan teknisen toteutuksen kannalta olennaisia. Käytännössä kolmannen osapuolen vastuulla on yksikkötestaus ja ne voitaisiin toteuttaa hyödyntämällä esimerkiksi Mocha JavaScript testausohjelmistoa, joka voi tuottaa monenlaisia testiraporttimuotoja (OpenJS Foundation 2023). Tosin sanoen, aiemmin esitetyn CI/CD prosessikuvauksen (Kuva 21) vaihetta yksi ei teknisiltä osin esitetä.

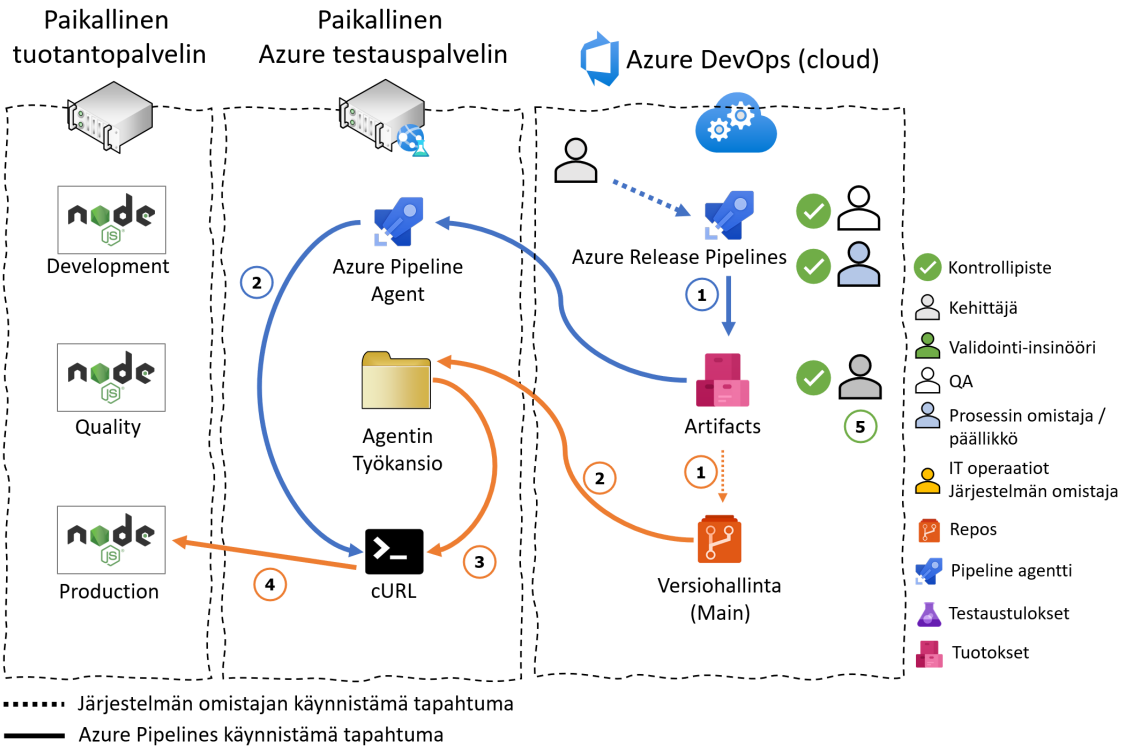
Kuvassa alla (Kuva 35) esitetään Azure Pipeline CI-työnkulku ja selostetaan numeroidut vaiheet olennaisilta osin. Kokoonpanossa laatu- ja tuotantojärjestelmät ovat validoituja ja vastaavasti kehitysympäristö ei ole, siitäkin huolimatta, että niitä ajetaan samalla palvelinalustalla.



Kuva 35. *Esitetyn CI-prosessin tekninen toteutus Azure Pipeline tuotteella.*

1. Järjestelmän omistajalle toimitetaan ohjelmakoodi ja testitapaukset paikalliseen versiohallintaan, jotka ”savutestataan” tilaajan kehitysympäristössä.
2. Järjestelmän omistaja lataa version versiohallintaan ja liittää pääprojektiin.
3. Azure käynnistää ao. versiohallintaan liitetyt Pipelinet.
4. Pipeline käynnistää paikallisella testipalvelimella olevan agentin ja lataa Azure versiohallinnasta testitapaukset ja lähdekoodin agentin työkansioon.
5. Agentti lataa lähdekoodin Node.js laatujärjestelmään (Quality) cURL komennolla.
6. Node.js järjestelmä käynnistyy automaattisesti, kun sille annetaan uusi ohjelma.
7. Agentti ajaa Newman sovelluksella työkansiossa olevat testitapaukset.
8. Newman sovellus tallentaa testien tuloksien JUnit XML-raportin työkansioon ja siirtää testitulokset Test Plans/Runs suorituksiin tarkastettavaksi.
9. Läpäistyjen testien jälkeen ohjelmakoodista versioidaan linkki Artifacts-arkistoon julkaisua varten, kun validointi-insinööri on vahvistanut validointiraportin, joka mm. edellyttää mahdollisten käsin tehtävien testien suorittamista hyväksytysti.

CD-prosessia kutsutaan Azure DevOpsissa Release Pipelineksi. CD-prosessi on teknisesti yksinkertaisempi (Kuva 36) kuin CI-prosessi ja kuvastaakin enemmän liiketoimintaprosessia kuin teknisiä vaiheita, koska siinä on enemmän manuaalisia kontrollipisteitä.

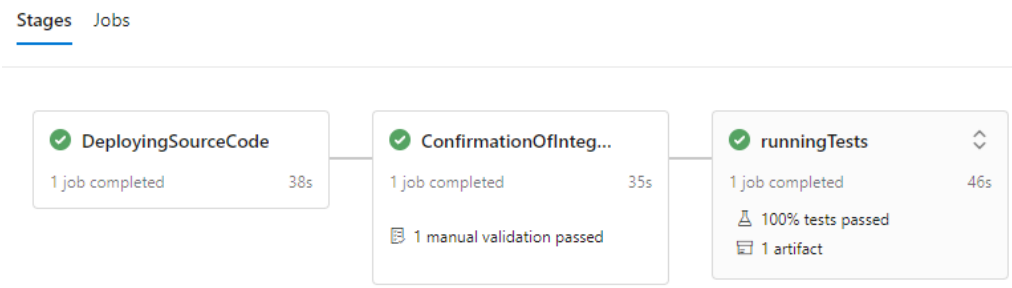


Kuva 36. *Esitetyn CD-prosessi toteutus Azure Release Pipeline tuotteella.*

1. Kun validointi on hyväksytty, käynnistetään Release Pipeline, joka lähettää pyynnöt luvalle edetä laatu- ja tuotanto-organisaation avainhenkilöille.
2. Azure Release Pipeline osoittaa oikean lähdekoodin (Artifacts) versiohallinnasta ja käynnistää paikallisella testipalvelimella agentin.
3. Azure Pipeline agentti suorittaa lähdekoodin sijoittamisen tuotantoympäristöön cURL kutsulla.
4. cURL kutsu kirjoittaa uuden lähdekoodin tuotantoympäristöön ja Node.js ympäristö käynnistyy uudelleen automaattisesti.
5. Lopuksi järjestelmän omistaja tarkastaa, että kaikki palvelut ovat käynnistyneet normaalisti. Tämä vaihe voidaan myös automatisoida luomalla Release Pipeline työnkulkuun suoritettavia testejä samaan tapaan kuin CI-prosessissa, jotka osoittavat, että sovellus on käynnissä ja normaalitilassa.

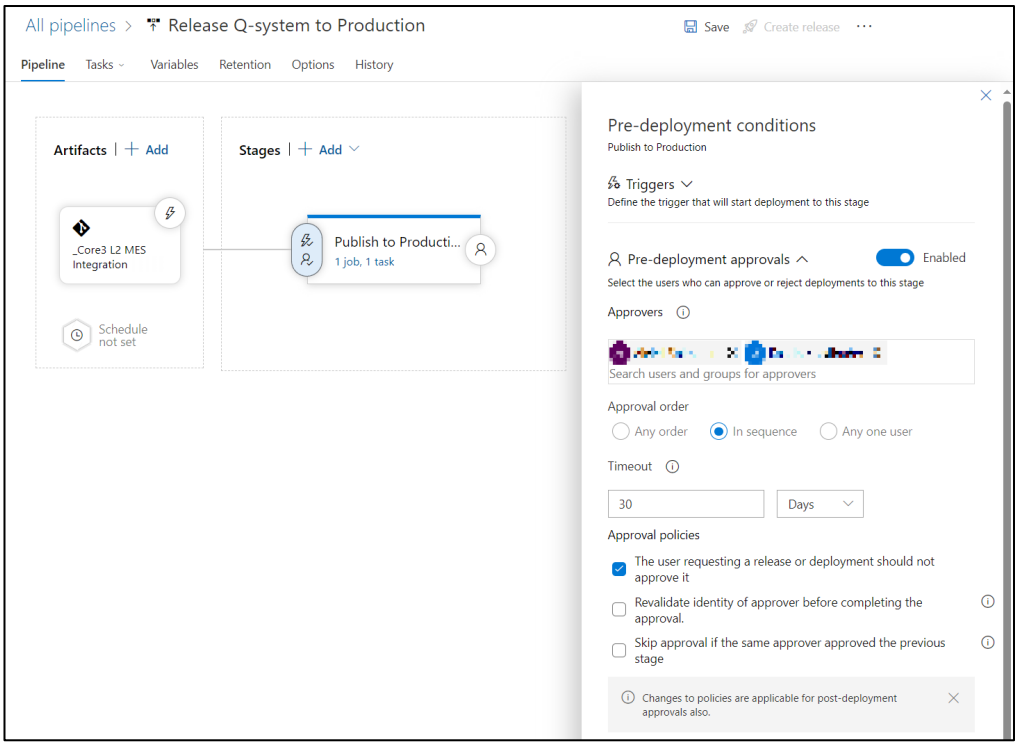
Azure DevOpsista löytyviä ”manual intervention” ja ”pre-deployment approvals” toiminnallisuuksia voidaan hyödyntää kontrollipisteiden luomiseksi. Manual intervention on CI

putkissa käytettävä vaihe, joka automaattisesti lähettää sähköpostia asianosaisille sidosryhmille määrityksen mukaan (Kuva 37), eikä etene seuraavaan vaiheeseen ennen kuin se on kirjattu tehdyksi.



Kuva 37. *Esimerkki Azure Pipeline vaiheesta, jossa keskellä manuaalinen tapahtuma (manual intervention). Manual intervention lähettää asianosaisille automaattisesti sähköpostin tarvittavista toimenpiteistä.*

CD putkissa (Release pipeline) vastaavasti jokaiselle vaiheelle on mahdollista määrittää käyttöönottoa edeltäviä vaatimuksia ja niihin liittyviä käytäntöjä (Kuva 38), kuten tässä tapauksessa eri sidosryhmiltä vaadittuja hyväksyntöjä.



Kuva 38. *Azure release pipelineen on mahdollista asettaa käyttöönottoa edeltäviä vaatimuksia, kuten hyväksynnät sidosryhmiltä.*

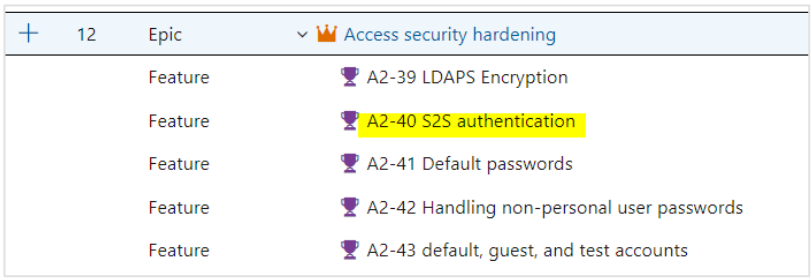
GMP:n kannalta on hyvä, että käytännöillä voidaan esimerkiksi estää julkaisijan itsensä hyväksymästä käynnistämänsä julkaisu, sekä se, että hyväksyntää suorittavaa käyttäjää pyydetään uudelleen tunnistautumaan (ks. 2.5 Tietokoneistetulta järjestelmältä vaadittavat ominaisuudet).

10.6 Automaattisten testitapauksien jäljitys

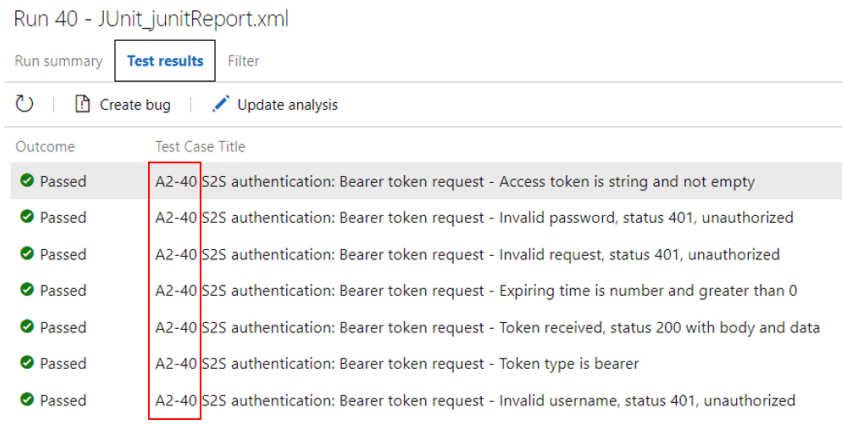
GMP edellyttää, että testitapaukset ovat jäljitettävissä vaatimukseen (6.1.1 Testitapausten jäljitettävyys vaatimuksiin). Edellä esitetyillä työkaluilla Azure DevOps ympäristössä ei pystytä yksittäisiä testitapauksia linkittämään suoraan vaatimukseen, koska testitapaukset ovat Postman scriptejä ja niistä toimitetaan raportti. Täten raportin yksittäisiä testitapauksia ei ole kovin helppo teknisesti linkittää muihin Azure työosioihin. Tällöin pitää testien jäljittämiseksi määritellä testien nimeämiskäytännöt, jossa linkitystieto on sisällytetty osaksi testitapauksen nimeämistä. Kuvissa 39, 40 ja 41 on esitetty, miten jäljitystiedot voitaisiin lisätä.

A2-40	System-to-external system, and vice versa, communication must support user ID and password or certificate-based authentication.	☒	☐	See comment on A2-38

Kuva 39. *Ote alkuperäisestä URS dokumentista, jossa yksilöivä vaatimustunniste A2-40.*



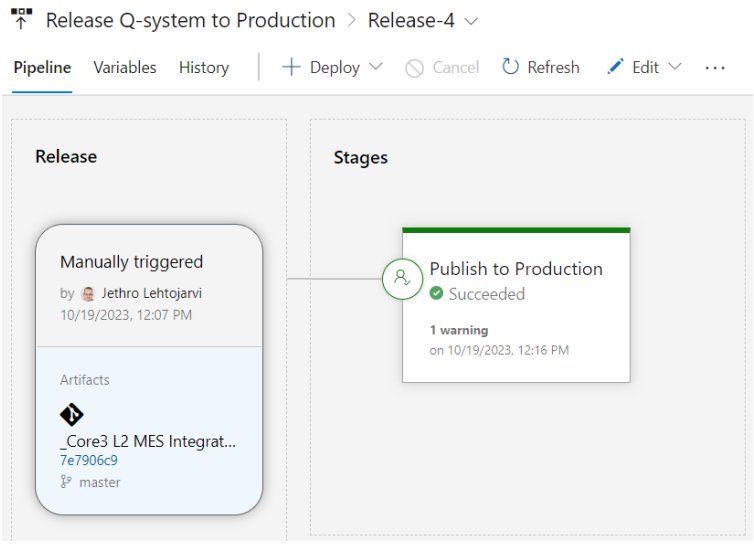
Kuva 40. *Sama vaatimus kirjattuna DevOps ominaisuuspinoon, jossa on vaatimuksen tunnistetieto.*



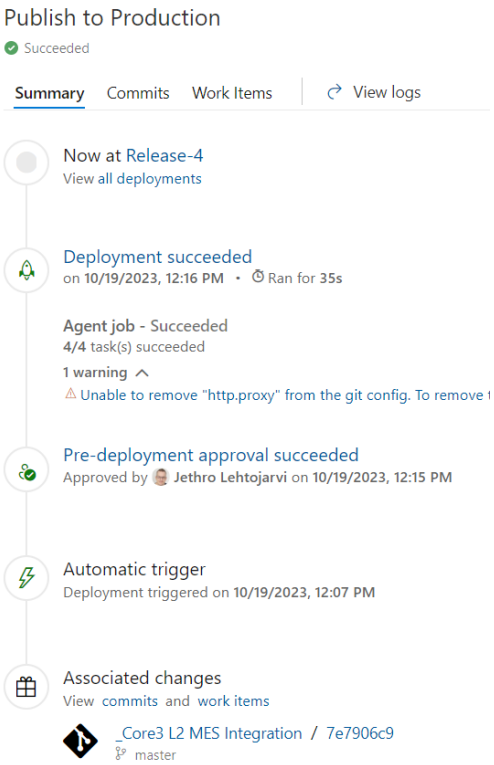
Kuva 41. *Yksittäisessä testipinossa näkyvä tunniste jäljitystä varten.*

10.7 Julkaistun version jäljitys

Release Pipeline tallentaa julkaisun yhteydessä käytetyn tuotoksen (artifact) yksilöllisen tunnisteen, versiohallinnan haaran, henkilön kuka on julkaisun käynnistänyt. Näiden lisäksi, nähtävillä on, että onko julkaisu työnkulun kannalta onnistunut ja milloin se on onnistunut, sekä sisältyykö julkaisuun joitakin varoituksia (Kuva 42).



Kuva 42. *Azure Relese Pipeline, jossa näkyy mm. mitä tuotosta on julkaisussa käytetty, kuka on julkaisun käynnistänyt ja milloin.*



Kuva 43. *Julkaisun tarkempi kirjausketju, josta on nähtävillä aikaleimat, tapahtumat ja tekijät.*

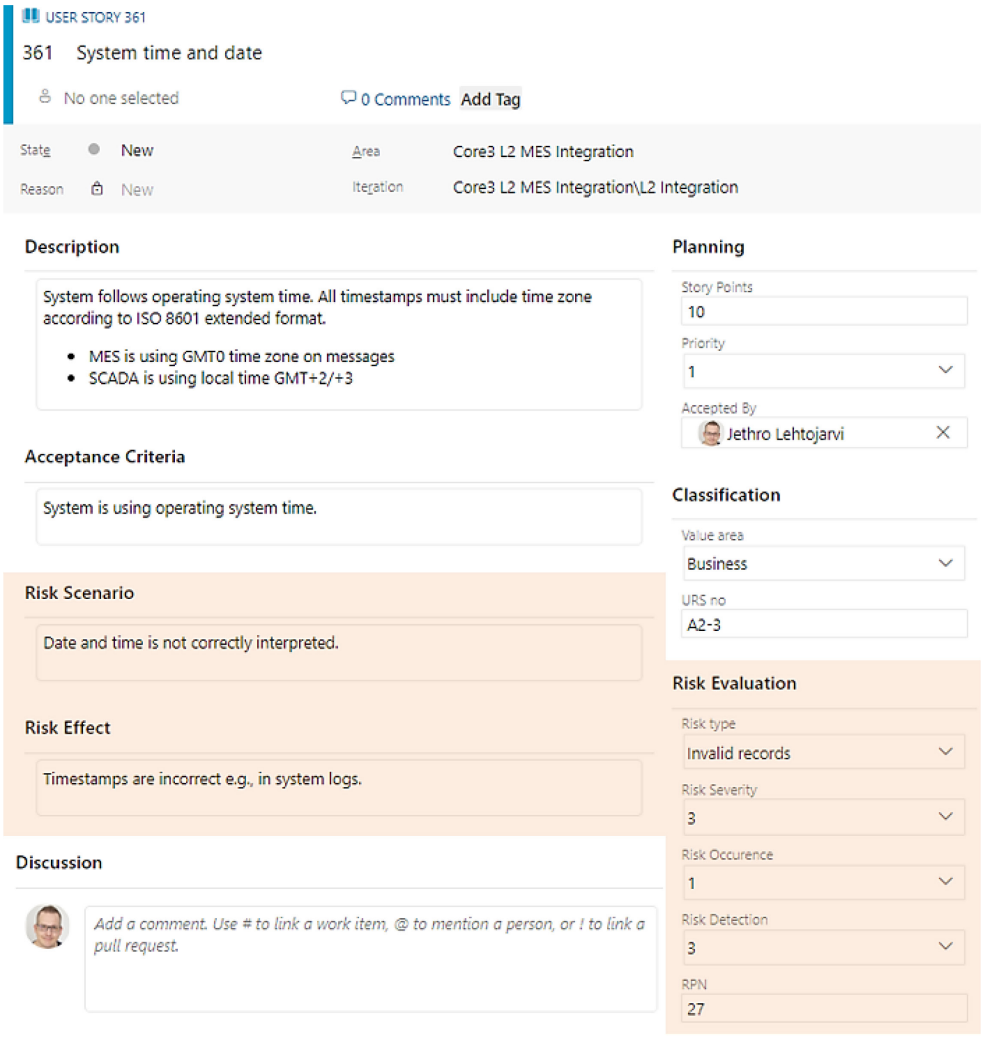
Julkaisun tapahtumien tarkempaa aikajanaa mahdollista tarkastella lähemmin (Kuva 43). Tarkemmasta tapahtumaketjusta löytyvät julkaisuun liittyvät hyväksynnät ja vaiheiden tarkemmat aikaleimat, joka täyttää aukottoman kirjausketjun vaatimukset, eli mitä, milloin ja kenen toimesta.

Esitetty julkaisuputki on tässä käyttötapauksessa hyvin suppea, mutta Release Pipeline voi koostua useista tuotoksista, kohteista ja vaiheista. Julkaisua voisi laajentaa esimerkiksi julkaisun päätteeksi ajettavista testitapauksista, jotka varmistavat, että julkaistu palvelu on tosiarvoisesti käynnissä ja palvelut ovat vastaavat kyselyihin odotetulla tavalla.

10.8 Riskien hallinta ja arviointi

GMP:n kannalta on olennaista, että suoritetut testitapaukset ovat jäljitettävissä vaatimuksiin (ks. kappale 6.1.1), josta esitettiin mahdollinen toimintatapa kappaleessa 10.6. Samat vaatimukset koskevat julkaistua versiota, joka on jäljitettävissä kappaleessa 10.7 esitetyllä tavalla. Edellä mainittujen jäljitysvaatimusten takia, tehdyt riskiarvot tulee myös olla jäljitettävissä käyttäjävaatimukseen, muutoinhan tarvittavia potilasturvallisuuteen liittyviä testitapauksia ei voitaisi määrittää riskilähtöisesti.

Riskienhallinnalliset toimenpiteet hallitaan tilaajaorganisaation toisessa järjestelmässä (kappale 7), mutta jäljittämisen kannalta tilanne on pirstaleinen. Azure DevOps mahdollistaa myös omien työosioiden (work item) määrittämisen, jos esimerkiksi halutaan entisestään granuloida tehtäviä pienempiin osiin. Seuraavassa kuvassa (Kuva 44) on ehdotus, miten riskienhallinnalliset (QRM) toimenpiteet voitaisiin liittää osaksi käyttäjätarinaa. Riskien arviointi käyttäjätarinan yhteydessä sujuvoittaisi riskien tunnistamista ja arviointia, jolloin erillistä prosessia riskien käsittelemiseksi ei tarvitsisi enää järjestää. Azure DevOps mahdollistaa seurannan kyselyiden avulla ja puutteisiin voitaisiin tarttua tapauskohtaisesti. Esitettyyn malliin on lainattu tilaajan kvalifiointitoimenpiteissä käytettyä vika- ja vaikutusanalyysimenetelmää (Soinio Hanna 2023), jonka avulla lasketaan riskeille riskin merkittävyyttä kuvaavia riskilukuja (RPN, risk priority number).



Kuva 44. *Käyttäjätarinaan lisätty kentät QRM prosessia varten (oranssilla korostetut osat).*

Mikäli riskiarvio tuottaa liian suuren riskiluvun, esimerkiksi suurempi kuin 9, tulee aloittaa suunnitelmat riskin pienentämiseksi siedettävälle tasolle. Kuvissa esiintyvissä RPN kentissä (Kuva 44, Kuva 45, Kuva 46) on käytetty tilaajan käytössä RPN laskennan kaavaa (1).

$$RPN = S^2 \cdot O \cdot D \qquad (1)$$

Missä

RPN = riskiluku

S = Riskin vakavuus

O = Riskin todennäköisyys

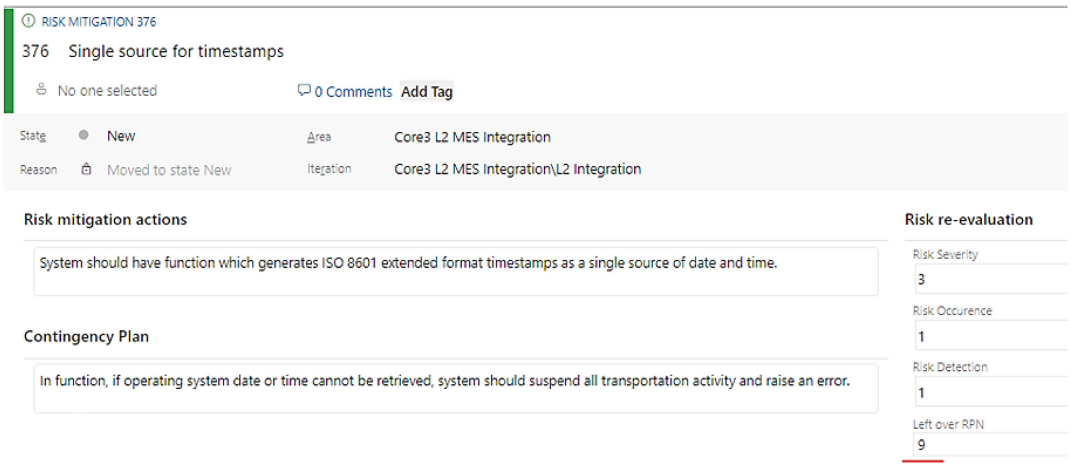
D = Riskin havaittavuus

Tämä menetelmä eroaa kappaleessa 7.1 (Tilaajan riskiarvioinnit) esitetystä siinä, että RPN luku kertoo esimerkiksi sen, että kuinka matalista tai keskitasoisista riskeistä on kyse, ts. kuinka lähellä seuraavaa tasoa ollaan ilman että eri arviointikriteerejä tarvitsee

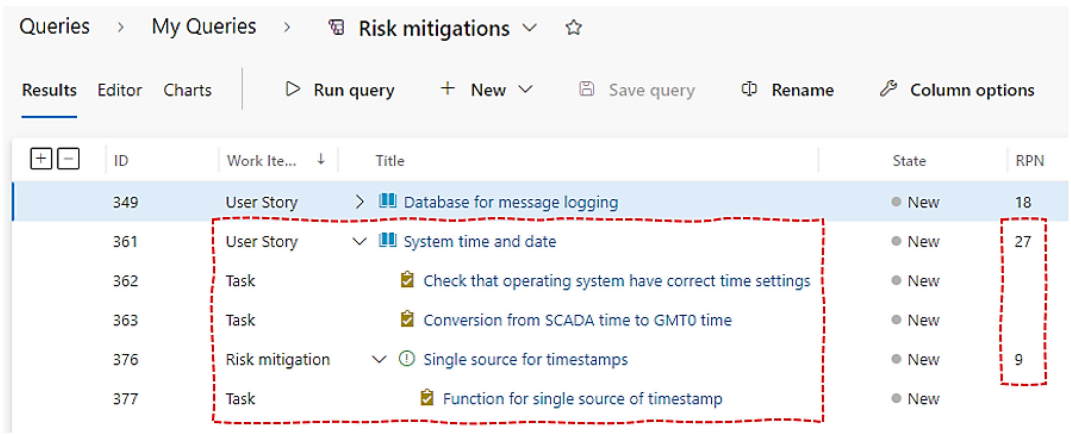
tulkita erikseen. Lisäksi numeerisen arvon käyttö mahdollistaa monipuolisimpien mittarien ja lajittelujen käytön.

Kappaleessa 5.3 (Laaturiskien hallinta) todetaan ensisijaisiksi keinoiksi liiketoimintaprosessin tai ohjelmiston muuttaminen. Lisäksi kappaleessa 7 (Tilaajan validointitestaus käytännössä) kuvataan lyhyesti, että eri vaatimuksien, spesifikaatioiden ja riskienhallinnallisten suhteiden ylläpitäminen on manuaalista, jota toteutetaan erillisellä riskienhallintadokumentilla.

Tätä varten, tietomalliin voitaisiin lisätä riskin mitikointiin (Risk mitigation) liittyvä työosio, josta on esimerkki seuraavassa kuvassa (Kuva 45), jonka yhteydessä hyödynnetään Azure DevOpsin ominaisuuttaa liittää työosioiden välille riippuvuussuhteita (Kuva 46).



Kuva 45. *Suunnitelma riskin pienentämiseksi siedettävälle tasolle, jolla RPN luku on saatu tasolle 9 (siedettävä taso), alkuperäisestä luvusta 27.*

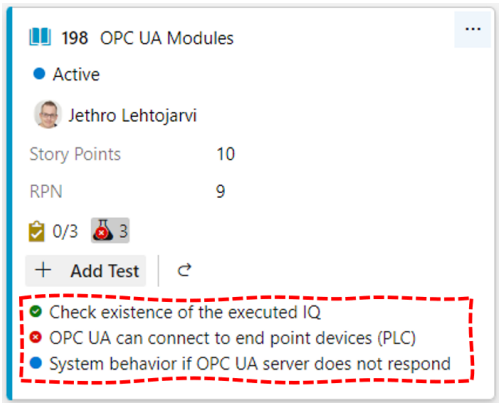


Kuva 46. *Riippuvuussuhteita sisältävät työosiot ovat linkitetty toisiinsa, jolloin RPN luvun madaltuminen siedettävälle tasolle on eri työosioiden suorittamisen seurauksena jäljitettävissä.*

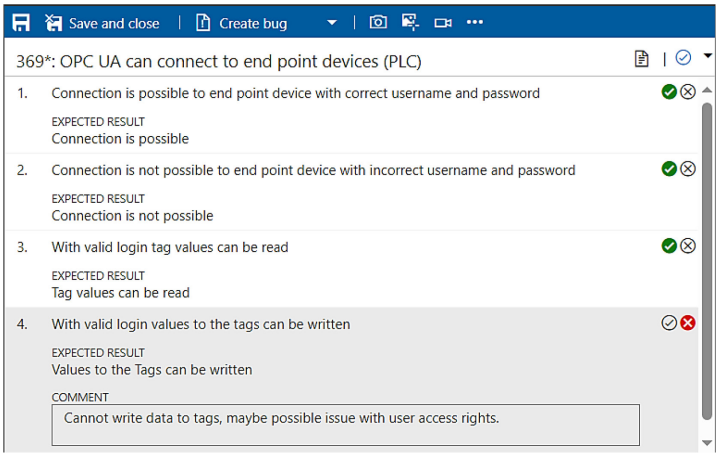
Edellä esitetyn esimerkin avulla jäljittäminen ja lisätöiden määrittäminen ohjelmiston muuttamiseksi onnistuu riskienhallinnallisista näkökulmista periytyen. Järjestely käytännössä automatisoi jäljittämiseen liittyvät muodollisuudet, koska tietomalli linkittää työosiot loogisesti toisiinsa ja niihin voidaan kohdistaa erilaisia hakuja ja automaattisia käytäntöjä. Tämä edellyttää, että käyttäjät toteuttavat järjestelyä oikein, mutta tilanne on toteuttamisen kannalta samankaltainen erillisillä dokumenteilla toteutettunakin.

10.9 Käyttäjävaatimustestaukset (UAT)

Erillisten testauspöytäkirjojen korvaamiseksi Azure DevOps ympäristössä on Test Plans ominaisuudet, jolla on mahdollista suunnitella käsikirjoitettuja testitapauksia suoraan ao. käyttäjätarinaan, ks. Kuva 47. Testitapaukset sisältävä eri askelia ja niihin määriteltyjä odotettuja tuloksia, jonka perusteella arvioidaan tulokseksi tosi/epätosi. Testien aikana testipöytäkirjalle (Test Runner, Kuva 48) voidaan lisätä kommentteja, sekä on mahdollista liittää kuvia, kuvakaappauksia, sekä tarvittaessa tallentaa nauhoitteita.

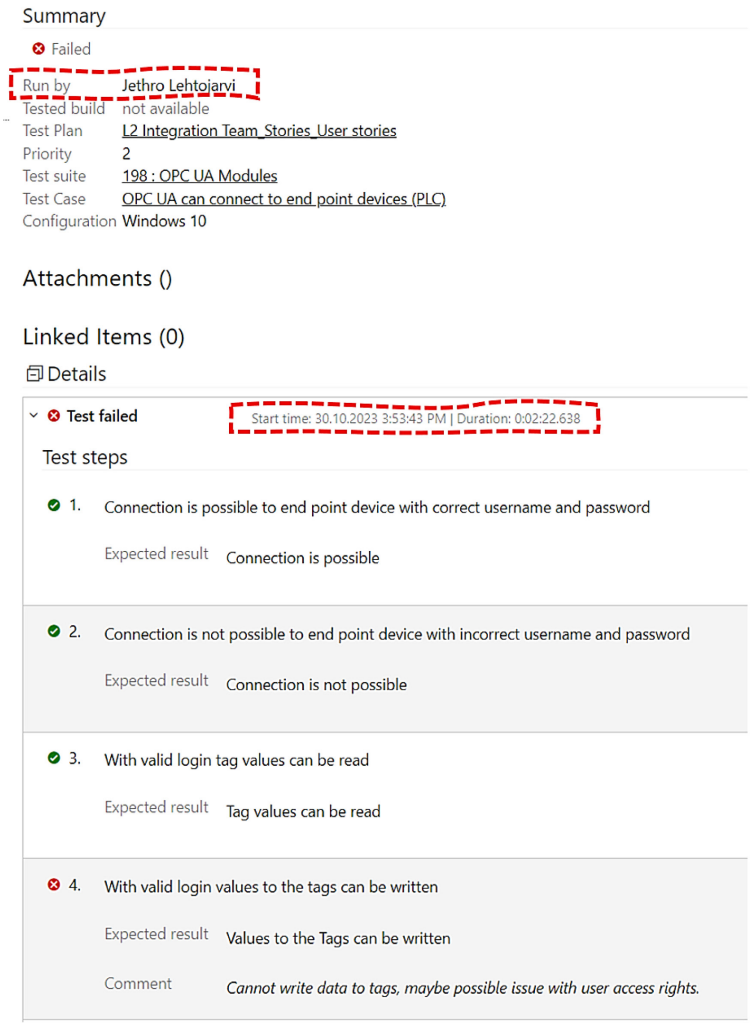


Kuva 47. *Käyttäjätarinaan liitettyjä testitapauksia.*



Kuva 48. *Testien suorittaminen (Azure Test Runner), johon voidaan tuloksien lisäksi liittää kuvia ja/tai nauhoitteita.*

Vakiokokoonpanolla testipöytäkirjaan ei raportoidu yksittäisten askelien suorittamisajakohtia, mutta tuloksien tarkempi tarkastelu osoittaa, että tulokset sisältävät tiedot testitapauksien suorittajasta, ajankohdan aloituksesta ja kokonaiskeston (Kuva 49). Käyttäjähyväksyntätestaukset (UAT) ovat tilaajan sääntelyn piiriin kuuluvaa toimintaa, joten vaatimus tietojen oikea-aikaisuudesta pitää täyttää (ISPE 2nd 2022, 359).



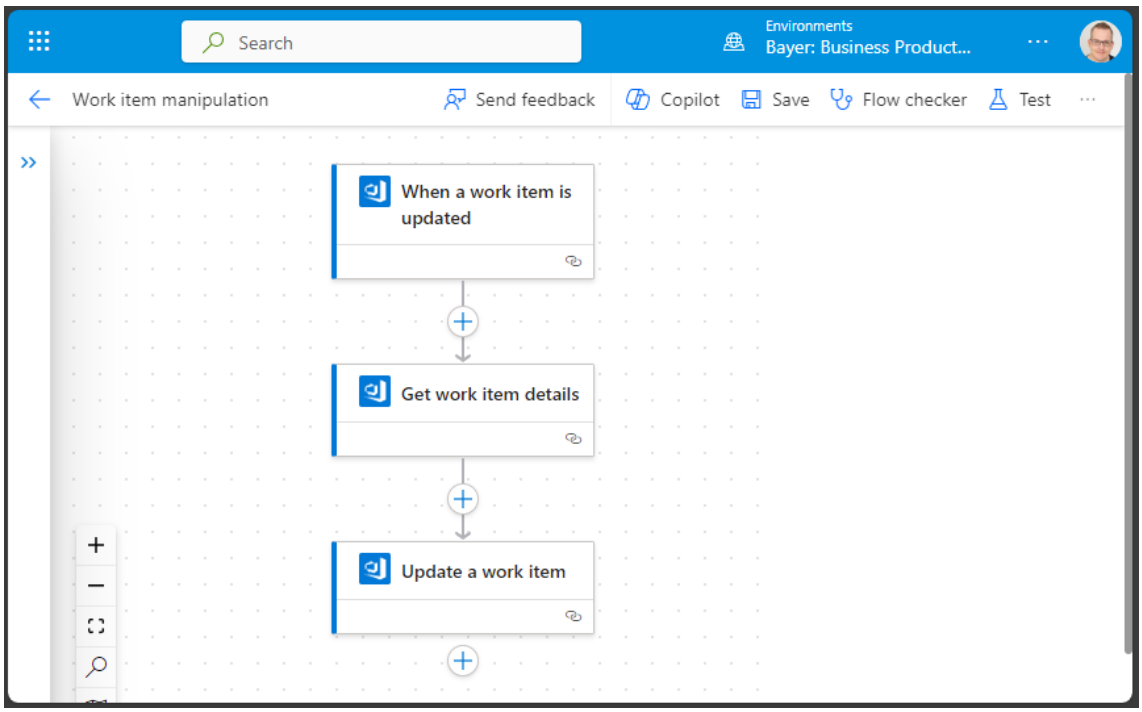
Kuva 49. *Testiyhteenvedosta näkyy suorittaja, aloitusaika ja kesto mutta ei yksittäisten vaiheiden aikaleimoja.*

Oikea-aikaisuusvaatimuksen täyttyminen vaatisi, että aikaleimat ovat nähtävillä erikseen rivi riviltä jokaisen yksittäisen suorituksen osalta, joten Test Runnerin tuottamat testipöytäkirjat eivät vaatimusta täytä ilman yksityiskohtaisia aikaleimoja. Puute ei kuitenkaan estä Test Runnerin käyttöä muun testauksen yhteydessä, koska toimittajat eivät ole sääntelyn piirissä toimivia yrityksiä ja ao. testaukset ovat usein toimittajan vastuulla. Puutteen korjaaminen vaatisi paikallisesti asennetun (on-premise) Azure DevOps ympäristön käyttöönottoa, jossa on tuki laajemmille konfigurointimahdollisuuksille (Import and export a Hosted XML process 2023). Tätä ominaisuutta ei työn aikana testattu, eikä todennettu, mutta on-premise ratkaisu voi tuoda muita ongelmia, esimerkiksi

integrointien ja liitettävyyden osalta. Ongelma ei ole kuitenkaan kovin keskeinen, koska tilaaja haluaa toteuttaa käsin tehtävät manuaaliset käyttäjähyväksyntätestitapaukset varsinaisessa validointijärjestelmässään.

10.10 Toiminnallisuuden laajentaminen ja integrointi

Azure DevOps toiminnallisuutta voidaan edelleen kehittää automatisoimalla tapahtumia Power Automate alustalla (Azure DevOps connectors n.d.), jolloin esimerkiksi erilaiset laskennat ja käyttäjätarinoiden tilatiedot voidaan säätää automaattisesti. Seuraavassa kuvassa on Power Automate alustalla oleva työnkulku (Kuva 50), joka laskee työosiolla olevan RPN-arvon, aina kun työosiosta tallennetaan uusi versio.



Kuva 50. *Power Automate alustalla toteutettu työosion tietojen päivitys.*

Power Automate laajentaa Azure DevOps toiminnallisuudet järjestelmää laajemmaksi, jonka avulla voidaan määritellä työosioille automaattisia toimenpiteitä esimerkiksi tilatietojen osalta tai estää käyttäjätarinan eteneminen toteutukseen, mikäli riskienhallinnallisissa toimenpiteissä on puutteita. Toisaalta Power Automate:n mukaan tuominen lisää toteutuksen kompleksisuutta ja lisää Azure DevOps ympäristön riippuvuutta muihin alustoihin, jolloin se voi olla häiriöherkempi.

11. TULOKSET

Viranomaiset käsittelevät tietokoneilla tuettuja lääketuotantoon liittyviä toimintoja laajemmin kuin vain tietojärjestelmätasolla. Tietojärjestelmän lisäksi laajuuteen kuuluu tuetut prosessit, toimintaohjeet, järjestelmää käyttävät ihmiset, sekä tietokone- ja tuotantolaitteistot. Edellä mainittujen seikkojen yhtälönä syntyy käsite tietokoneistettu järjestelmä. Tietokoneistettuiin järjestelmien hankintaan liittyen viranomaisten keskeinen ohjaava arvo on potilasturvallisuus ja siihen liittyvät riskit ja niiden riittävä hallinta, jota toteutetaan hyvien tuotantotapojen (GMP) viranomaisohjeiden mukaisesti. ISPE:n GAMP 5 oppaassa vastaavasti kannustetaan vahvaan vuorovaikutukseen yrityksen sisäisten sidosryhmien ja toimittajan sidosryhmien kanssa tietokoneistettujen järjestelmien käyttöönotoissa. Se ei kuitenkaan ole standardi, vaan pyrkii käytännönläheisesti kuvaamaan tarvittavia toimenpiteitä, jotta sääntelyn vaatimat kriteerit täyttyvät

Viranomaismääräykset määrittelevät lähtökohtaisesti mitä tuloksia tarvitaan, jotta määräykset ja lain säätelemät pykälät täyttyvät, mutta eivät ota varsinaisesti kantaa siihen, miten ratkaisut tulisi toteuttaa tai miten ratkaisuarkkitehtuureja tulisi arvioida teknisesti tai liiketoiminnan näkökulmista. Näiden vaatimuksien muuttaminen teknisiksi ratkaisuiksi jää toteuttavien organisaatioiden tulkattavaksi ja viimekädessä loppukäyttäjän validointiprosessin arvioitavaksi siitä, että soveltuuko toteutus aiottuun käyttöön potilasriskiarviointeihin pohjautuen. Viranomaiset ja ISPE ovat kuitenkin tunnistaneet, että sovellusteollisuus tuottaa enenevässä määrin ratkaisuja ketteriin menetelmiin nojautuen myös säädellyissä ympäristöissä ja ovatkin ajanmukaistaneet ohjeitaan ja suosituksiaan vastaamaan uusien kehittyvien teknologioiden jalkautumiseen. Loppukädessä, sääntelyn piirissä toimivan yrityksen tulee parhaansa mukaan tulkita viranomaisohjeita ja päättää sen perusteella minkälaisia hallinnollisia toimenpiteitä se käyttää liiketoimintaprosesseissaan. Tietokoneistettujen järjestelmien osalta kyse on elinkaarimallista, joka käsittää järjestelmän koko elinkaaren hankinnasta lakkautukseen asti ja kaiken näiden väliltä, kuten varsinaisen käytön ja käytön aikaiset valvontatoimenpiteet.

Viranomaisnäkökulmasta DevOps toimintamallin käyttöön ei ole esteitä elinkaaren hallinnan välineenä. Perinteiseen lääketeollisuuden käyttämään spesifikaatioverifikaatio elinkaarimalliin verrattuna terminologiat, vaiheet ja toimintatavat eroavat selkeästi, jotka mahdollisesti voivat johtaa erilaisuudessaan tulkinta- ja ymmärryseroihin toimintaa harjoittavassa yrityksessä eri sidosryhmien välillä. Käyttöönottoon pitää sisällyttää riittävät koulutustoimenpiteet.

DevOps toimintamalli pyrkii paloittelemaan vaatimukset yksittäisiksi työnkuluiksi, jossa prosessin kaikki vaiheet toteutetaan yksittäisen vaatimuksen tasolla, jonka lopputuloksena syntyy ketteryys. DevOps toimintaa soveltavilta organisaatioilta vaaditaan uudenlaista ajattelua ja eri sidosryhmiltä vaaditaan entistä toistuvampaa osallistumista elinkaaren kaikkiin vaiheisiin, joka on DevOps toiminnan keskeinen olemus. Vaatimuksien paloittelu ”miniprosesseiksi” vaatii varta vasten tähän tarkoitukseen suunnitellun ja konfiguroidun työkalusovelluksen käyttöä, jotta kokonaisuuksia pystytään hallitsemaan tehokkaasti ja mitään unohtamatta. Ilman kunnollisia työkaluja DevOps toimintamallin toteuttaminen voi osoittautua hankalaksi, ellei jopa mahdottomaksi.

Testausautomaatio on luonteva osa DevOps toimintamallia, jota toteutetaan testaus- ja toimitusputkien (CI/CD) avulla. Testausautomaation käyttö ei teknisellä tasolla varsinaisesti vaadi DevOps toimintamallia, mutta ilman työssä esitettyjä DevOps käytäntöjä testien tuottamien tuloksien liittäminen vaatimuksiin jäljitystä varten GMP:n mukaisesti voi olla hankala järjestää. GMP:n kannalta testausautomaation tuottamat tulokset ovat samanarvoisia manuaalisten testauksien tuottamien tuloksien kanssa, kunhan testausautomaatiolla tuotetaan samalla tavalla tulkittavia vastauksia. GMP:n kannalta testausautomaatio tunnistetaan kyvykkääksi erityisesti muutoshallintojen yhteydessä suoritettavien regressiotestauksien yhteydessä, jonka etuna on suorituskyvyn tasaisuus ja nopeus.

Esitetty tekninen toteutus osoittaa, että Azure DevOps on kyvykäs työkalu, sen laajat konfigurointiominaisuudet helpottavat järjestelmän sovittamista erilaisiin liiketoimintatarpeisiin. Se täyttää sellaisenaan merkittäviltä osin GMP toiminnan edellyttämiä toimintoja, mutta se edellyttää tietynlaisia käytäntöjä, kuten vaatimuksien numerointia manuaalisesti ja erilaisten manuaalisten kontrollipisteiden lisäämistä toimitusputkiin, joiden käyttö on ohjeistettava. GMP:n kannalta merkittävimmät puutteet ovat oikea-aikaisuuden esittämisessä manuaaliseen testaamiseen käytettävien pöytäkirjojen osalta. Azure DevOps sovelluksen luonteen vuoksi erilaiset raportit ja koosteet ovat käytännössä verkkosivuja, jotka rajoittavat mahdollisuuksia tulostaa muodollisia raportteja varsinaisen validointidokumentaation liitteiksi. Kyselyiden avulla voidaan tuottaa raporttien kaltaisia koosteita, ja ne toimivat johtamisen kannalta oivallisesti työkalun eri sivustonäkymissä. Kyselyiden tuottamien listauksia tulostamiseksi, esimerkiksi PDF muotoisina tiedostoina, ei ole varsinaisesti ratkaisua valmiina, joka hankaloittaa validointiraportin laatimista.

Työn tuloksina osoitettiin, että Azure DevOps pilvipalvelulla voidaan ajaa validoitavan integraatioratkaisun rajapintoihin automaattisia testitapauksia paikallisessa tuotantoympäristössä hyödyntämällä testaustyökaluja. Testit tuottavat Azure DevOps yhteensopivia raportteja, joita voidaan hyödyntää GMP-toiminnassa. Työssä esiteltyjen testaus- ja toimitusputkien määritteleminen on melko suoraviivaista, mutta teknisesti monimutkaisemmat putket vaativat perehtymistä YAML-konfigurointikieleen. Testaus ja toimitusputkiin on mahdollista liittää GMP:n vaatimuksienmukaisia kontrollipisteitä ja kaikki tapahtumat ovat jäljitettävissä niihin liittyviin tuotoksiin. Työn tuloksina esitettiin laajennettu Azure Agile tietomalli, johon oli liitetty tilaajan käyttämä vaihtoehtoinen riskienhallintaprosessi osaksi käyttäjätarinaa. Laajennus vähentää riskienhallinnallisiin toiminnallisuuksiin liittyviä pirstaleisuuksia, mutta tämä edellyttää kattavan raportin tulostamista järjestelmästä validointiraportin liitteeksi. Varsinaisen validointiraportin generointia automaattisesti ei tutkittu, mutta se voisi olla yksi tulevaisuuden kehityshaara.

DevOps toimintamallin jalkauttamiseksi tarvitaan myös uudenlaista osaamista esimerkiksi automaattisten testaus- ja toimitusputkien määrittelemiseksi ja konfiguroimiseksi. Putkien laatiminen on käyttötapauskohtaista ja aihe on tekniseltä olemukseltaan sellainen, että sitä ei voida generalisoida kuin periaatteellisella tasolla, koska toteutukset voivat erota hyvinkin paljon toisistaan. Generalisointi tarkoittaa GMP:n mukaisten kontrollipisteiden lisäämistä osaksi putkien työnkulkua. Osaamistarvetta tai roolin määrittelyä erilaisten putkien laatimiseksi ei tullut esille käsitellyissä viranomaisohjeissa tai oppaissa. DevOps insinööri voisi olla tehtävänimike, joka voisi suorittaa edellä kuvattuja tehtäviä ja voisi toimia ikään kuin DevOps lähettiläänä kehitystiimien ja muiden sidosryhmien välillä varsinaisen DevOps kehittämisen lisäksi.

Azure DevOps työkalun toiminnallisuutta on mahdollista kehittää automatisoimalla tapahtumia Power Automate alustalla, jolloin esimerkiksi erilaiset laskennat ja käyttäjätarinoiden tilatiedot voidaan säätää automaattisesti, perustuen vaikka riskinhallinnallisten tilannetietojen mukaan. Automatiikalla voidaan esimerkiksi estää käyttäjätarinan vienti toteutukseen, mikäli sillä on liian korkea riskiluku. Näiden lisäksi muihin edellä mainittuihin puutteisiin, kuten raportit, voisi tutkia ratkaisumahdollisuuksia esimerkiksi Power Automate toteutuksilla.

12. YHTEENVETO JA PÄÄTELMÄT

Tutkimuksessa pyrittiin etsimään edellytyksiä DevOps toimintamallin käyttöönottoon lääketeollisissa toimintaympäristöissä, sekä arvioida voidaanko DevOps toimintamalliin olennaisena osana kuuluvaa testausautomaatiota hyödyntää validointityön tukemiseen järjestelmäintegraatiosovelluksen yhteydessä.

Työn toteutus vaati perehtymistä viranomaisohjeisiin, ISPE GAMP 5 oppaaseen ja tilaajan käytäntöihin tietokoneistettujen järjestelmien hankintaan ja validointiin liittyen. Toimintamallin osalta työ vaati perehtymistä myös ohjelmistotuotannon DevOps toimintamalliin liittyviin keskeisiin periaatteisiin, sekä ohjelmistoteollisuuden testausautomaatiokäsitteisiin ja prosesseihin. Teknisen toteutuksen osalta tutkittiin Azure DevOps, Postman ja Newman työkaluohjelmien kyvykkyyksiä, sekä mahdollisuuksia soveltaa niillä testausautomaatiota järjestelmäintegraatioratkaisun REST API rajapintojen validointitestauksen tukena. Työssä esitetään yleinen GMP:tä noudatteleva DevOps elinkaariprosessi ja sitä sovelletaan esimerkin omaisesti Azure DevOps työkaluohjelmistolla testaus- ja toimitusputkilla toteutettuna järjestelmäintegraation elinkaariprosessin tukena. Esimerkissä syntyneiden testausautomaation tuottamien tuloksien soveltuvuutta arvioidaan validointityön tukemiseen, jolle ei varsinaisesti nähty estettä, mutta se vaatii lisäkäytäntöjä GMP:n mukaisten jäljitysvaatimuksien toteutumiseksi.

Viranomaismääräysten ja ohjeiden kannalta ketterä kehittäminen ja DevOps tyyppinen toimintamalli voi olla osa säädellyn liiketoiminnan prosesseja, se on täysin liiketoimintaa harjoittavan yrityksen itsensä päätettävissä, mutta se on myös päätettävä, koska DevOps ei ole standardi, jota noudattaa. DevOps vaatii lääketeollisuuden hyvin konservatiivisesta validointilähestymistavasta luopumista ja soveltuvien työkaluohjelmistojen käyttöönottoa, kouluttamista ja uuden omaksumista. Ketterät menetelmät soveltuvat parhaiten sellaisiin toteutuksiin, joihin on oletettavissa liiketoiminnan kannalta toistuvia muutoshallintoja, kuten järjestelmäintegraatiot ja nousevat teknologiat, kuten tekoäly- ja data-analytiikkateknologiat. Konservatiivinen lähestymistapa voi olla liiketoiminnan kannalta liian hidas näillä uusilla teknologia-alueilla, erityisesti kehittyneillä data-analytiikka alueella.

Tilaajan käytössä oleva Azure DevOps työkaluohjelmisto soveltuu tietyin varauksin tilaajan GMP:n mukaiseen ketterään kehittämiseen ja tarvittavien tilaus- ja toimitusputkien toteuttamiseen ja se voidaan laajentaa toteuttamaan myös riskien hallinnallisten

toimenpiteiden suorittamista. Esitetty ratkaisu kykenee jäljittämään ohjelmiston käyttäjävaatimukset, niihin liittyvät testaustulokset ja sovelluksien eri kehitysversiot aukottomasti toisiinsa, sekä julkaisuun liittyviin vaiheisiin on liitettävissä GMP:n vaatimia laatukontrolleja. Azure DevOps ei sovellu kokonaisvaltaisesti kaikkiin GMP:n mukaisiin testaamistarpeisiin sellaisenaan ja toteutusta on laajennettava muilla käytännöillä. Keskeiset puutteet kohdistuvat käyttäjähyväksyntätestaustapauksien suorittamiseen liittyviin aikaleimoihin, joita ei yksittäisten testisuoritusten kohdalla tallenneta, tämä on selkeä GMP puute. Testausta voidaan kuitenkin soveltaa muihin testaustarpeisiin, kuten yksikkö- ja integraatiotestaus, jos ne tuottavat yhteensopivia raportteja. Johtamisen kannalta kyselytyökalu eri Azure DevOps työosioiden ja niiden suhteiden visualisointiin on oivallinen, mutta raporttitulosteiden saaminen niistä validointiraportin tueksi voi olla teknisesti hankalaa. Tätä ei työssä teknisellä tasolla tarkemmin tutkittu.

Tutkimuksen tulokset ovat tilaajalle merkityksellisiä, koska se onnistui vastaamaan tutkimuskysymyksiin, joiden asetannalla pyrittiin arvioimaan ketterien menetelmien ja testausautomaation hyödyntämistä tilaajan GMP-toiminnassa. Tilaaja oli tunnistanut, että ketterille menetelmille voisi olla tarvetta, mutta niiden käyttöä ei ollut vielä tutkittu käytännön tasolla. Tutkimuksen tulokset tukevat näiden työkalujen käyttöönottoa erityisesti uusien hankkeiden elinkaarimalleiksi, erityisesti nousevien teknologioiden käyttöönotoissa. Toteutuksien monimutkaistuessa, myös testaus monimutkaistuu. Testauksien suorittaminen kokonaan manuaalisesti voi osoittautua aivan liian hitaaksi lähestymistavaksi erityisesti käyttöönoton jälkeisissä muutoshallinnoissa. Tästä syystä tilaajalla on tahtoa kehittää nousevien teknologioiden tueksi myös niihin liittyvien validointitoimenpiteiden toteuttamista.

Työn aihepiirejä hieman laajemmin katseltuna ja toisistaan irrotettuna, työssäni ei varsinaisesti tuoda mitään uutta, mutta näiden kahden maailman, DevOps ja GMP, yhdistäminen niin käytännön kuin terminologian osalta on sellainen yhdistelmä, jota ei kovinkaan paljon tämän työn parissa tullut esille. Tälle lienee syynä se, että lääketeollisuus on toimintaprosesseissaan hyvin konservatiivinen ja hitaasti kehittyvä, koska nämä toimialat seuraavat hyvin pitkälti ISPE:n julkaisemia oppaita, joista ensimmäinen painos on vuodelta 2008 ja toinen 2022. Näistä jälkimmäisessä vasta puhutaan DevOps toimintamalleista ja ketteryydestä, ja niistäkin tämän tutkimuksen perusteella varsin pinnallisesti.

Työtä voisi jatkaa perehtymällä syvällisemmin Azure DevOps alustan mukauttamiseen GMP puuteiden paikkaamiseksi, mutta sitä voisi hyödyntää myös liiketoiminnan eri tarpeisiin, kuten projektityöskentelyyn esimerkiksi tuotantoautomaatioratkaisujen parissa,

ei siis pelkästään sovellustuotannon tarpeisiin. Tämän lisäksi olisi mielenkiintoista tutkia miten lääketeollisuus ylipäätään suhtautuu ketteriin ja DevOps toimintamalleihin ja kuinka yleistä näiden käyttö on, mitä hankaluuksia ovat kohdanneet ja mitä etuja on havaittu, sekä miten viranomaiset ovat tarkastuksissa näihin suhtautuneet?

Työ sai alkunsa ideasta pyrkiä helpottamaan validointityötä digitalisaation tuoman lisääntyvän kompleksisuuden ja järjestelmien välisten tietovirtojen integrointitarpeiden lisääntyessä. Eniten aikaa kului työn sopivan aihekokonaisuuden hahmottamisessa, koska, kaikki mitä lisäksi tämän aiheen ympärillä tapahtui, on osa päivätyötäni. Aihekokonaisuus oli mielestäni vaativa. Ajankäytöllisistä haasteista huolimatta työn tuloksia pidän erittäin onnistuneina, koska työn tulokset avaavat ovet validointityön kehittämiselle ja tarjoavat myös mahdollisen vaihtoehdon perinteiselle lääketeollisuuden V-mallille konkreettisilla käytännön tavoilla. Esitetyt tekniset ratkaisut eivät täytä kaikkia tarpeita, mutta tämä on lopputuloksena hyvä alku, joten työn tuloksia voisi kutsua kokonaisuutena GxP DevOps MVP:ksi.

LÄHTEET

21 CFR. (1997). Code of Federal regulations Title 21 Part 11 Electronic Records; Electronic Signatures. Verkkosivu. Viitattu 18.9.2022. Saatavilla: https://www.ecfr.gov/current/title-21/chapter-I/subchapter-A/part-11

About ISPE. (n.d.). ISPE. Verkkosivu. Viitattu 29.9.2022. https://ispe.org/about

Agile Alliance. (n.d.). What is the Agile Manifesto? Verkkosivu. Viitattu 2.6.2023. Saatavilla: https://www.agilealliance.org/agile101/the-agile-manifesto/

Agile Model Tutorialspoint. (n.d.). Tutorialspoint. Verkkosivu. Viitattu 23.10.2022. Saatavilla: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

Azure DevOps connectors (n.d.). Microsoft Learn. Verkkosivu. Viitattu 1.11.2023. Saatavilla: https://learn.microsoft.com/en-us/connectors/visualstudioteamservices/

Courtemanche, M. Mell, E. Gills, A. (n.d.). What is DevOps? The ultimate guide. n.d. Verkkoartikkeli. Viitattu 22.3.2023. Saatavilla: https://www.techtarget.com/searchitoperations/definition/DevOps

Doyle, A. (15.3.2022). Critical Thinking Definition, Skills, and Examples. Artikkeli. Viitattu 15.3.2023. Saatavilla: https://www.thoughtco.com/critical-thinking-definition-with-examples-2063745

EduQuest. (2011). Comparison of FDA's Part 11 and the EU's Annex 11. Pdf-dokumentti. Viitattu 18.9.2022. Saatavilla: http://www.eduquest.net/Advisories/Comparison%20of%20FDA%20Part%2011%20and%20EU%20Annex%2011.pdf

EMA. (2022). Human regulatory, Good Manufacturing practice. Viranomaisohje. Viitattu 18.9.2022. Saatavilla: https://www.ema.europa.eu/en/human-regulatory/research-development/compliance/good-manufacturing-practice

EMA. (n.d.). Management Board members. Viranomaislähde. Viitattu 18.9.2022. Saatavilla: https://www.ema.europa.eu/en/about-us/who-we-are/management-board/members

EudraLex Volume 4 Good Manufacturing Practice Annex 11. (30.6.2011). Computerised Systems. Viranomaisohje. Viitattu 18.9.2022. Saatavilla: https://health.ec.europa.eu/system/files/2016-11/annex11_01-2011_en_0.pdf

EudraLex Volume 4 Good Manufacturing Practice Chapter 1. (23.1.2013). Viranomaisohje. Viitattu 17.6.2023. Saatavilla: https://health.ec.europa.eu/system/files/2016-11/vol4-chap1_2013-01_en_0.pdf

EudraLex Volume 4 Good Manufacturing Practice Chapter 6. (1.10.2014). Viranomaisohje. Viitattu 19.6.2023. Saatavilla: https://health.ec.europa.eu/system/files/2016-11/2014-11_vol4_chapter_6_0.pdf

EudraLex Volume 4 Good Manufacturing Practice: Annex 15. (1.10.2015). Qualification and Validation. Viranomaisohje. Viitattu 2.6.2023. Saatavilla: https://health.ec.europa.eu/document/download/7c6c5b3c-4902-46ea-b7ab-7608682fb68d_en?filename=2015-10_annex15.pdf

EudraLex Volume 4. (n.d.). Good Manufacturing Practice guidelines. Viranomaisohje. Viitattu 21.10.2022. Saatavilla: https://health.ec.europa.eu/medicinal-products/eudralex/eudralex-volume-4_en

FDA. (2022). Code of Federal Regulations Title 21, Chapter 1, Subchapter A, Part 7, § 7.1 Scope. Viranomaisohje. Viitattu 18.9.2022. Saatavilla: https://www.ecfr.gov/current/title-21

Fimea Tietoa Fimeasta. (n.d.). Verkkosivu. Viitattu 18.9.2022. Saatavilla: https://www.fimea.fi/tietoa_fimeasta

Fimea Valvonta. (n.d.). Verkkosivu. Viitattu 18.9.2022. Saatavilla: https://www.fimea.fi/valvonta

Fimea. (2022). Määräys FIMEA/2022/000806 Lääkkeiden hyvät tuotantotavat sekä kliinisissä lääketutkimuksissa käytettävien lääkkeiden valmistusta koskevat vaatimukset. Viranomaisohje. Viitattu 18.9.2022. Saatavilla: https://www.fimea.fi/documents/160140/764653/M%C3%A4%C3%A4r%C3%A4ys+6+2022+FI.pdf/b4b3f280-20a3-2f53-ac97-3b12f5f4e8fb?t=1656673756242

Gambier Sara. (4.10.2023). Industrial IT Manager. Bayer Nordic SE. Keskustelu. Turku.

Glossary ISPE. (n.d.). ISPE. Verkkosivu. Viitattu 29.9.2022. Saatavilla: https://ispe.org/glossary?title_contains=GAMP&langcode=en

Guidance Documents ISPE. (n.d.). ISPE. Verkkosivu. Viitattu 29.9.2022. Saatavilla: https://ispe.org/publications/guidance-documents

ICH Q9. (2021). ICH guideline Q9 (R1) on quality risk management (EMA/CHMP/ICH/24235/2006) 16.12.2021. Viranomaisohje. Viitattu 7.1.2023. Saatavilla: https://www.ema.europa.eu/en/documents/scientific-guideline/international-conference-harmonisation-technical-requirements-registration-pharmaceuticals-human-use_en-3.pdf

Import and export a Hosted XML process (25.2.2023). Microsoft Learn. Verkkosivu. Viitattu: 1.11.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/organizations/settings/work/import-process/import-process?view=azure-devops

ISPE 2nd. (2022). GAMP 5 A Risk-Based Approach to Compliant GxP Computerized Systems Second Edition. ISPE. 2022. ISBN 978-1-946964-57-1.

ISPE. (2008). GAMP 5 A Risk-Based Approach to Compliant GxP Computerized Systems. ISBN 1-931879-77-X.

Kansalliskirjasto. (2018). Tietotermit, Tietojärjestelmä. Verkkosivu. Viitattu 18.9.2022. http://urn.fi/URN:NBN:fi:au:tt:t79

Kasurinen, J. (2013). Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo. ISBN EPUB 978-952-291-102-5.

Körber Pharma, (n.d.). Werum PAS-X MSI Plug & Produce: Fast, seamless, integrated. Verkkosivu. Viitattu: 6.4.2023. Saatavilla: https://www.koerber-pharma.com/en/solutions/software/werum-pas-x-add-ons/werum-pas-x-plug-produce

Lwakatare, L. (2017). DevOps adoption and implementation in software development practice. Tampere: Juvenes print. ISBN 978-952-62-1711-6

Marciniak J. (2001). Process Models in Software Engineering. Pdf-dokumentti. Viitattu 20.9.2022. Saatavilla: https://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf

Microsoft Learn. (3.5.2023). About default processes and process templates. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops&tabs=agile-process

Mojtaba, S., Muhammand, A. B., Liming, Z. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE access, 2017, Vol.5, Article 7884954. p.3909-3943. Saatavilla: https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=7884954

OpenJS Foundation. (23.4.2023). Mocha Reporters. Verkkosivu. Viitattu 24.10.2023. Saatavilla: https://mochajs.org/#reporters

Packer, D. (2.2.2022). Continuous Integration vs. Continuous Delivery vs. Continuous Deployment. Verkkosivu. Viitattu 6.6.2023. Saatavilla: https://www.plutora.com/blog/continuous-integration-continuous-delivery-continuous-deployment

Pavlović Alexander. (2020). What is FDA 21 CFR Part 11 European Equivalent? Ideagen. Verkkosivu. Viitattu 18.9.2022. Saatavilla: https://www.ideagen.com/thought-leadership/blog/what-is-fda-21-cfr-part-11-european-equivalent.

PIC/S Secretariat. (2007). PIC/S Guidance PI 011-3. PIC/S. 20.9.2007. Viranomaisohje. Viitattu 20.9.2022. Saatavilla: https://picscheme.org/docview/3444

Pihlajamäki, J. (n.d.). Mitä on DevOps? Verkkoartikkeli. Viitattu 22.3.2023. Saatavilla: https://otaverkko.fi/mita-on-devops/

Postman. (n.d.). What is postman? Verkkosivu. Viitattu: 11.4.2023. Saatavilla: https://www.postman.com/

Prince, S. (11.2.2016). The Product Managers' Guide to Continuous Delivery and DevOps. mind the PRODUCT. Verkkosivu. Viitattu: 17.6.2023. Saatavilla: https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/

Publish test results to Azure Pipelines. (7.3.2023). Microsoft Learn. Verkkosivu. Viitattu: 16.4.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/reference/publish-test-results-v2?view=azure-pipelines&tabs=trx%2Ctrxattachments%2Cyaml

Ries, E. (2011). The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses. New York: Crown Books. eISBN: 978-0-307-88791-7

Running collections on the command line with Newman. (10.4.2023). Verkkosivu. Viitattu 11.4.2023. Saatavilla: https://learning.postman.com/docs/collections/using-newman-cli/command-line-integration-with-newman/

Self-hosted Windows agents. (27.1.2023). Microsoft Learn. Verkkosivu. Viitattu: 11.4.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=azure-devops

Soinio Hanna. (27.9.2022). Computer System Validation Lead. Bayer Oy. Keskustelu. Turku.

Soinio Hanna. (2.10.2023). Computer System Validation Lead. Bayer Oy. Keskustelu. Turku.

Spiral Model Tutorialspoint. (n.d.). Tutorialspoint. Verkkosivu. Viitattu 23.10.2022. Saatavilla: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

TMAP. (2019). FDA's Technology Modernization Action Plan (TMAP) 18.19.2021. Viranomaisjulkaisu. Viitattu 27.1.2023. Saatavilla: https://www.fda.gov/media/130883/download

Toivakka, H. & Granlund, T. & Poranen, T. & Zhang, Z. (2021). Towards RegOps: A DevOps Pipeline for Medical Device Software. Tutkimus. Viitattu 10.10.2023. Saatavilla: https://link.springer.com/chapter/10.1007/978-3-030-91452-3_20

Tähtinen, S. (2005). Järjestelmäintegraatio. Tarve, Vaihtoehdot, Toteutus. Helsinki: Talentum. ISBN 952-14-0854-5.

U.S Government. (2007). PUBLIC LAW 110–85—SEPT. 27, 2007, SEC. 222, REGISTRATION, (b), REGISTRATION OF FOREIGN ESTABLISHMENTS. Viranomaisohje. Viitattu 18.9.2022. Saatavilla: http://www.gpo.gov/fdsys/pkg/PLAW-110publ85/pdf/PLAW-110publ85.pdf

Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. Fifth International Conference on the Innovative Computing Technology. 22.5.2015. IEEE. Verkkosivu. Viitattu: 6.6.2023. Saatavilla: https://ieeexplore.ieee.org/xpl/conhome/7166535/proceeding

W3schools® of Technology. (2022). SDLC V-Model. W3schools® of Technology. Verkkosivu. Viitattu 20.10.2022 Saatavilla: https://www.w3schools.in/sdlc/v-model.

Waterfall Model Tutorialspoint. (n.d.). Tutorialspoint. Verkkosivu. Viitattu 23.10.2022. Saatavilla: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

What is Azure DevOps? (11.10.2022). Microsoft Learn. Verkkosivu. Viitattu: 11.4.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

What is Azure Pipelines? (4.1.2023). Microsoft Learn. Viitattu: 12.4.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

Writing tests 2023. Writing tests. (10.4.2023). Verkkosivu. Viitattu: 11.4.2023. Saatavilla: https://learning.postman.com/docs/writing-scripts/test-scripts/

YAML pipeline editor. (11.2023). Microsoft Learn. Verkkosivu. Viitattu: 16.4.2023. Saatavilla: https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/yaml-pipeline-editor?view=azure-devops