



TAMPEREEN TEKNILLINEN YLIOPISTO  
TAMPERE UNIVERSITY OF TECHNOLOGY

MATIAS KANGAS

RAUTATIESTANDARDIEN MUKAINEN DOKUMENTAATIO KET-  
TERÄSSÄ OHJELMISTOKEHITYKSESSÄ

Diplomityö

Tarkastaja: Assistant Professor  
David Hästbacka

Tarkastaja ja aihe hyväksytty  
28. Marraskuuta 2018

## TIIVISTELMÄ

**MATIAS KANGAS:** Rautatiestandardien mukainen dokumentaatio ketterässä ohjelmistokehityksessä

Tampereen teknillinen yliopisto

Diplomityö, 56 sivua, 34 liitesivua

Marraskuu 2018

Automaatiotekniikan koulutusohjelma

Pääaine: Automaatiotekniikka

Tarkastaja: Assistant Professor David Hästbacka

Avainsanat: dokumentaatio, ketterät menetelmät, rautatiestandardit, agile, scrum, xp, kanban, lean

Ohjelmistotuotannon kehittyessä myös sen ongelmat ovat kehittyneet. Alkuun ongelmia tuotti ohjelmistotyön kaavamaisuus, jäykkyys ja hidas reagointikyky virheiden ratkaisemiseksi. Näiden ongelmien ratkaisuun kehitettiin erilaisia ketteriä ohjelmistokehitystapoja, joissa irtaudutaan perinteisestä kaavamaisesta toimintatavasta vähentämällä dokumentaatiota ja keskittymällä jatkuvaan kehitystoimintaan.

Ketterät menetelmät toimivat hyvin menestyksekkäästi, mutta ongelmia tuottavat erityisesti joillakin tekniikan aloilla pakollinen dokumentaatio, sekä isoilla organisaatioilla tiedon ja osaamisen katoaminen liian vähäisen dokumentaation johdosta.

Diplomityössä pureudutaan ketterään kehitykseen rautatieteollisuudessa. Asiakasyrityksen dokumentaatiohaasteisiin ja pyritään löytämään keinoja toteuttaa yrityksen kannalta vaadittavaa dokumentaatiota ketterässä ohjelmistokehityksessä.

Työssä arvioidaan perinteisten ja ketterien menetelmien hyviä ja huonoja puolia, sekä dokumentaation vaikutusta koko organisaatioon. Mikäli yritys ei ole puhdas ohjelmointi firma niin kompromissi perinteisen ohjelmistokehityksen ja ketterien menetelmien välillä tuottaa koko yritystä hyödyttävää dokumentaatiota ja ohjelmistokehitykselle ketterän rakenteen.

## ABSTRACT

**MATIAS KANGAS:** Documentation in agile software development based on railway standards

Tampere University of Technology

Master of Science Thesis, 56 pages, 34 Appendix pages

November 2018

Master's Degree Programme in Automation Engineering

Major: Automation Engineering

Examiner: Assistant Professor David Hästbacka

Keywords: documentation, agile, railway standards, scrum, xp, Kanban, lean

Software development has evolved over the years. The changes software development has faced have not erased problems but changed them. At the beginning the cause of problems were formality, stiff and slow reaction to problem solving. Agile methods were invented to combat these issues. An agile approach detaches itself from traditional formal approach by reducing documentation and focusing on constant development.

Agile methods have worked very well but they cause problems especially in technical fields where documentation is mandatory. In big organisations minimal documentation cause a loss of knowledge and know-how.

This thesis focuses on agile software development in railway industry. The customer companys documentation challenges and attempts to find ways to produce documentation in agile software development that is useful to the whole company.

The thesis examines the benefits and drawbacks of tradition and agile methods as well as the effect of documentation to the whole organisation. If the company is not a pure software company the compromise in documentation between traditional and agile methods benefits the whole company and gives the software development agile structure.

## ALKUSANAT

Aluksi haluan kiittää diplomityöni tarkastajaa David Hästbackaa sekä ohjaani Tero Laihoa heidän tarjoamastaan tuesta diplomityöni kirjoitusprosessissa. Haluan myös kiittää Mitron Oy:n työntekijöitä heidän innokkuudestaan osallistua diplomityössä tehtyyn kyselyyn ja heidän antamistaan neuvoista. Suuri kiitos perheelle ja ystäville, joiden tuki ja motivaatio auttoivat koko prosessin läpi.

Tampereella, 1.4.2018

Matias Kangas

# SISÄLLYSLUETTELO

1.	JOHDANTO .....	1
1.1	Aiheen rajausta ja tavoitteet .....	2
1.2	Menetelmät .....	2
1.3	Rakenne .....	2
2.	OHJELMISTOKEHITYSMALLIT JA STANDARDIEN VAATIMUKSET .....	4
2.1	Yleiset dokumentaatiomallit .....	4
2.2	Ketterä ohjelmistokehitys .....	6
2.2.1	Scrum .....	8
2.2.2	EXTREME PROGRAMMING .....	10
2.2.3	Kanban .....	13
2.2.4	Lean .....	14
2.2.5	DevOps .....	16
2.3	Rautatiestandardien vaatimukset dokumentaatioon .....	20
2.3.1	IEEE 1558 .....	21
2.3.2	EN 50128 .....	23
2.4	Erot yleisten dokumentaatioääntöjen ja ketterän ideologian välillä .....	27
2.5	Ketterän kehityksen luomat haasteet dokumentaatioissa .....	28
3.	YRITYKSEN OHJELMISTODOKUMENTAATION HAASTEET .....	29
3.1	Nykyinen käytäntö .....	29
3.2	Haastattelun tuloksia .....	29
3.3	Ongelmat .....	33
3.4	Dokumentaation tarpeet .....	33
4.	TEORIAN SOVELTAMINEN KÄYTÄNTÖÖN .....	35
4.1	Toteutus ja tulokset .....	35
4.2	Polarion .....	35
4.2.1	Projektit .....	35
4.2.2	Tiedostojen tuonti ja vienti .....	35
4.2.3	Työkappaleet .....	37
4.2.4	Uudelleenkäytä ja haarauta .....	37
4.3	Case Study IPEP .....	38
4.3.1	SDD .....	38
4.3.2	SRS .....	38
4.3.3	STPr .....	38
4.3.4	Dokumentaatio .....	38
4.3.5	Polarion dokumenttien luomisessa .....	39
4.3.6	Polarion dokumenttien tuonnissa .....	39
4.3.7	Dokumenttien luominen työkappaleista .....	40
4.3.8	Vienti .....	41
4.4	Dokumenttipohjat .....	43
4.5	Yritysdokumentaation päivitys standardin mukaiseksi .....	43

4.6	Dokumentaatiohaasteiden ylittäminen .....	45
4.7	Ketterän dokumentaation toteuttaminen yrityksessä.....	47
4.8	Ketterien menetelmien tehostaminen .....	48
5.	TULOSTEN ARVIOINTI .....	50
5.1	Dokumentaation tarve ja hyödyt ketterässä kehityksessä .....	50
5.2	Dokumentaation arvo .....	50
5.3	Rautatiestandardien mukainen dokumentaatio ketterässä ohjelmistokehityksessä .....	51
6.	YHTEENVETO .....	53
	LÄHTEET .....	54
	LIITE A: HAASTATTELU LOMAKE.....	57
	LIITE B: SRS POHJA .....	62
	LIITE C: SDD POHJA .....	74
	LIITE D: SRTM POHJA .....	80
	LIITE E: STPR POHJA .....	86

## KUVALUETTELO

<b>Kuva 1.</b>	<i>Perinteinen vesiputousmalli [8] .....</i>	<i>6</i>
<b>Kuva 2.</b>	<i>Ketterien menetemien vapaus asteikko [6].....</i>	<i>7</i>
<b>Kuva 3.</b>	<i>Scrum projektin etenemiskaava [5].....</i>	<i>9</i>
<b>Kuva 4.</b>	<i>Extreme programming [5].....</i>	<i>12</i>
<b>Kuva 5.</b>	<i>Kanban [15] .....</i>	<i>13</i>
<b>Kuva 6.</b>	<i>Toyota Production System tukipilarit [17].....</i>	<i>14</i>
<b>Kuva 7.</b>	<i>Toyota tuotantojärjestelmä kolmio [17].....</i>	<i>15</i>
<b>Kuva 8.</b>	<i>Epäselvyyden seinä [24].....</i>	<i>17</i>
<b>Kuva 9.</b>	<i>DevOps tuotannon, testauksen ja toimituksen osalta [24].....</i>	<i>17</i>
<b>Kuva 10.</b>	<i>DevOpsin kolme tapaa. Ylin virtaus, keskimäinen palaute, alin oppiminen ja kokeilu [25] .....</i>	<i>18</i>
<b>Kuva 11.</b>	<i>DevOps:in perusajatus vasemmalle siirtymisestä [21] .....</i>	<i>20</i>
<b>Kuva 12.</b>	<i>SIL tasot ja tasojen kuvaukset [2] .....</i>	<i>23</i>
<b>Kuva 13.</b>	<i>EN50128 Ohjelmistokehityksen elinkaari versio 1 [2] .....</i>	<i>25</i>
<b>Kuva 14.</b>	<i>EN50128 Ohjelmistokehityksen elinkaari versio 2 [2] .....</i>	<i>26</i>
<b>Kuva 15.</b>	<i>Ohjelmistokehityksen elinkaari SIL0 dokumentaatio suosituksilla, muokattu lähteestä [2].....</i>	<i>27</i>
<b>Kuva 16.</b>	<i>SRS dokumentin tuominen (engl. import) ja järjestelmä vaatimusten (engl. system requirement) automaattinen luominen.....</i>	<i>36</i>
<b>Kuva 17.</b>	<i>Polarion:iin tuotu dokumentti otsikko tunnisteella (yläpuolella) ja automaattisella järjestelmä vaatimusten luomisella (alapuolella) .....</i>	<i>37</i>
<b>Kuva 18.</b>	<i>SRS dokumentti Word:ssä ja Polarion:ssa.....</i>	<i>39</i>
<b>Kuva 19.</b>	<i>Tuonti ja automaattinen työkappaleiden luonti.....</i>	<i>40</i>
<b>Kuva 20.</b>	<i>SRS dokumentista tehty system requirement, otsikolla ja kommenttiosilla sekä vain kommenttiosiollla .....</i>	<i>40</i>
<b>Kuva 21.</b>	<i>Työkappaleista luotu dokumentti.....</i>	<i>41</i>
<b>Kuva 22.</b>	<i>Word dokumentin Export toiminto valintoineen.....</i>	<i>42</i>
<b>Kuva 23.</b>	<i>ReqIF Export valinnat .....</i>	<i>42</i>
<b>Kuva 24.</b>	<i>Reuse toiminto .....</i>	<i>43</i>
<b>Kuva 25.</b>	<i>Työkappale valikko ja dokumenttiin tuodut vaatimukset.....</i>	<i>44</i>
<b>Kuva 26.</b>	<i>Siirto komennon valintaikkuna .....</i>	<i>44</i>
<b>Kuva 27.</b>	<i>XP-malli ja SIL0 dokumentaatio vaatimukset, muokattu lähteestä [5].....</i>	<i>46</i>
<b>Kuva 28.</b>	<i>Scrum malli ja SIL0 dokumentaatio vaatimukset, muokattu lähteestä [5].....</i>	<i>47</i>
<b>Kuva 29.</b>	<i>Kanban ketterässä ohjelmistokehityksessä, muokattu lähteestä [15] .....</i>	<i>48</i>

## LYHENTEET JA MERKINNÄT

CDRL	Contract Data Requirements List on lista hyväksytyistä datavaatimuksista tietyille osastolle. Se on osa suurempaa sopimusta.
EN	European Standard ovat dokumentteja, jotka yksi kolmesta Euroopan standardiorganisaatiosta (CEN, CENELEC, ETSI) on hyväksynyt.
IEEE	Institute of Electrical and Electronics Engineers kansainvälinen tekniikan järjestö, joka kehittää standardeja ja julkaisee useita tieteellisiä julkaisuja
ReqIF	Requirements Interchange Format on XML-tiedostoformaatti, jota käytetään vaatimusten ja metadatan lähettämiseen ja vastaanottamiseen ohjelmistojen välillä.
SCI	Software Configuration Item(s) on kooste laitteesta, ohjelmistosta tai molemmista, joka toimii yksittäisenä kappaleena konfiguraation hallintaa prosessissa.
SDD	Software Design Description IEEE 1558 standardissa määritelty dokumentti, jossa ohjelmiston malli kuvataan.
SIL	Safety Integrity Level EN50128 Standardin määrittelemä turvallisuuden taso.
SRS	Software Requirements Specification IEEE 1558 standardissa määritelty dokumentti, johon kirjataan ohjelmiston vaatimukset.
STPr	Software Test Procedure IEEE 1558 standardissa määritelty dokumentti, joka kuvaa testausvälineet ja kuinka testaus tehdään.
SRTM	Software Requirement Traceability Matrix IEEE 1558 standardissa määritelty dokumentti, jossa kerrotaan mistä dokumentista vaatimukset ovat lähtöisin, mihin ohjelmistosuunnitteluun niitä käytettiin ja mikä on ohjelmiston testaus dokumentti
TPS	Toyota Production System Toyotan kehittämä tuotantojärjestelmä, jonka pohjalta syntyi Lean.
XML	Extensible Markup Language on dokumenttien koodaamiseen tarkoitettu kieli, jota sekä ihminen että kone voi lukea.
XP	eXtreme programming Ketterä ohjelmistotuotannon menetelmä.

# 1. JOHDANTO

Rautatieteellisyydessä toiminta perustuu kattavaan dokumentaatioon, jotta rautatieliikenne voidaan taata turvalliseksi. Kaikki rautatiedokumentaatio perustuu rautatiestandardissa esitettyihin vaatimuksiin, joiden avulla eri tahojen dokumentaatio täyttää saman formaatin ja mahdollistaa yritysten välisen helpon kommunikaation. Dokumentaatio ei ole vain valmistajan vastuu, vaan monesti tärkeä osa tuotetta. Tästä syystä joillakin yrityksillä on omia lisävaateita dokumentaatioon liittyen. Rautatieteellisyys on erittäin iso kokonaisuus, joka sisältää paljon eri tuotekategorioiden erikoistuneita yrityksiä. Näille yrityksille on tärkeää tietää, mitkä osat laajasta standardoinnista ovat heille oleellisia. [1,2,3]

Ketterän ohjelmistokehityksen ydin ajatuksena on minimoida dokumentaatiota ja antaa ohjelmistokehittäjille vapaammat kädet ohjelmoimiseen. Tässä törmätään ristiriitaiseen tilanteeseen, jossa tuotantotavassa pyritään minimaaliseen dokumentaatioon, mutta asiakas ja standardit vaativat kattavaa dokumentaatiota. [4,5]

Diplomityössä tutkitaan, kuinka standardien mukaista ohjelmistodokumentaatiota hyödynnetään ketterässä ohjelmistokehityksessä. Yleisellä tasolla ohjelmistodokumentaatiotavat ja -vaatimukset ovat tarkasti standardoituja, mutta ketterässä ohjelmistokehityksessä ei ole dokumentaatiosta mitään standardeja, vain yleisiä ohjeita. Diplomityössä perehdytään rautatiestandardien näkökulmasta ketterän ohjelmistokehityksen dokumentaatiotarpeisiin ja toteutustapoihin.

Tutkimuksen kannalta keskeisiksi kysymyksiksi nousevat:

- Dokumentaation tarve ja hyödyt ketterässä kehityksessä?
- Onko dokumentaatio siihen käytetyn ajan arvoista?
- Mitä on rautatiestandardien mukainen dokumentaatio ketterässä ohjelmistokehityksessä?
- Kuinka rautatiestandardien mukainen dokumentaatio ja ketterä ohjelmistokehitys voidaan toteuttaa kohdeyrityksessä?

Diplomityön keskeisenä osana on selvittää minkälaisia vaatimuksia rautatiestandardit asettavat Mitron Oy:n dokumentaatiolle sekä tutkia, kuinka dokumentaatiota tulisi luoda ketterässä ohjelmistokehityksessä.

## 1.1 Aiheen rajausta ja tavoitteet

Tämä diplomityö on Mitron Oy:n toimeenpanema tutkimus, jolla pyritään ratkaisemaan yrityksen dokumentaatioissa ilmeneviä haasteita. Yrityksen ohjelmistotuotanto pyrkii seuraamaan ketteränkehityksen oppeja, mutta rautatiestandardit sekä asiakasvaatimukset luovat eriäviä vaatimuksia dokumentaation suhteen. Edellä mainittujen parametrien mukaan aihe rajautuu yrityksen kannalta kahteen tärkeimpään rautatiestandardiin, EN 50128 ja IEEE 1558. Standardit käsittelevät rautatieteellisyydessä käytettyjen ohjelmistojen sekä ohjelmistokehitystyön dokumentaatio vaatimuksia. Asiakasvaatimusten osalta perehdytään haastatteluiden, sekä kolmen erityyppisen asiakkaan vaatiman dokumentin näkökulmasta aiheeseen. Yleisen teorian osalta tutkitaan perinteistä ohjelmistodokumentaatiota ja ketteriä menetelmiä muutamien keskeisten ratkaisuiden osalta. [1,2,3]

Ketteristä menetelmistä on paljon oman aihealueen tutkimuksia, jotka pääasiassa keskittyvät selvittämään mitä ketterät menetelmät ovat, kuinka niitä tulisi hyödyntää ja minkälaisissa organisaatioissa ketterät menetelmät ovat tuoneet positiivisia vaikutuksia. Ketterien menetelmien eräs ominaisuus on lähes olematon dokumentaatio. Aiempi tutkimus ei ole täsmällisesti tarkastellut, kuinka ketterät menetelmät pärjäävät suurempaa dokumentaatiomäärää vaativassa organisaatioissa.

## 1.2 Menetelmät

Tutkimustyö aloitettiin aiheeseen tutustumisella ja materiaalin keräämisellä. Aiheen teoreettisuuden vuoksi aineistoon perehtyminen on avainasemassa yrityksen nykytilan ja kehityssuunnan selvittämisessä sekä parannuskohtien tunnistamisessa.

Yrityksen kannalta nykytilan selvittäminen oli tärkeää, jotta voidaan määrittää missä ollaan ja mihin mennään. Nykytilaa selvitettiin haastattelemalla työntekijöitä muun muassa siitä, kuinka nykyisin dokumentoidaan ja miten se vaikuttaa heidän työhönsä. Haastattelukysymyksistä tehtiin kirjallinen versio, joka annettiin haastateltavalle haastattelun alussa (Liite A). Keskustelu nauhoitettiin ja vastaukset kirjattiin pääkohdin ylös haastattelulomakkeeseen. Haastattelussa saatujen vastausten pohjalta tehtiin taulukko, jonka sarake sisältää keskeisen toteamuksen ja kuinka vastaajien antamat vastaukset skaaloutuivat.

Työssä kokeillaan dokumentaation luontia dokumentointiin erikoistuneella ohjelmalla, minkä pohjalta tutkitaan, kuinka modernien dokumentaatiota helpottavien ohjelmien avulla ketterä kehitys sujuu tavallista suuremmalla dokumentaatiotarpeella.

## 1.3 Rakenne

Työ rakentuu kuudesta luvusta, joita ovat johdanto, ohjelmistokehitysmallit ja standardien vaatimukset, yrityksen ohjelmistodokumentaation haasteet, teorian soveltaminen käytäntöön, tulosten arviointi ja yhteenveto.

- Luku 1 johdanto kertoo suurpiirteisesti mitä työssä käsitellään, mitä pyritään saavuttamaan ja minkälaisilla menetelmillä tuloksia pyritään tuottamaan.
- Luvussa 2 käsitellään ohjelmistokehitysmalleja perinteisestä vesiputousmallista erilaisiin ketteriin menetelmiin ja vertaillaan niiden välisiä eroja ja haasteita sekä rautatiestandardien asettamia vaatimuksia ohjelmistodokumentaatiolle.
- Luku 3 käsittelee kohdeyrityksen nykytilaa ja haasteita, joita selvitettiin työntekijöitä haastatteleamalla.
- Luvussa 4 esitellään projektinjohtamista helpottavaa työkalua ja esitellään käytännön ratkaisutapoja työssä esille nouseviin ongelmiin ja kohdeyrityksen ketterän ohjelmistokehityksen tehostamiseen.
- Luvussa 5 arvioidaan työssä saavutettujen tulosten vaikutusta kohdeyritykseen.
- Luvun 6 yhteenvedossa tiivistetään työn oleellinen sisältö.

## 2. OHJELMISTOKEHITYSMALLIT JA STANDARDIEN VAATIMUKSET

Tässä luvussa käsitellään ohjelmistojen dokumentaatiota yleisellä tasolla sekä ketterän ohjelmistokehityksen ja rautatiestandardien luomia vaatimuksia dokumentaatioon.

### 2.1 Yleiset dokumentaatiomallit

Ohjelmistodokumentaation määrä ja laajuus vaihtelee rajusti yrityksen koon ja tarpeiden mukaan. Teoreettista oppia ja jaottelua dokumentaatiosta on paljon, eivätkä suurin osa julkaisuista ole yksimielisiä edes dokumenttien nimistä. Julkaisut, kuten I. Haikkalan ja J. Märijärven Ohjelmistotuotanto [4], Henrik Knibergin Kanban vs Scrum [6] ja Ian Sommervillen Software Documentation [7], ovat kuitenkin samaa mieltä siitä, että dokumentaation määrä vaihtelee projektin koosta riippuen muutamasta dokumentista täysiin arkistokaappeihin. Ohjelmistodokumentaation laiminlyöminen tai tekemättä jättäminen johtaa ongelmiin niin projektin seurannan, johtamisen, asiakasesittelyn kuin käytön kannalta. Virheet dokumentaatioissa johtavat väärinymmärryksiin, jotka hidastavat prosessin kulkua ja johtavat tuotetta käytettäessä odottamattomiin virheisiin. Erityisesti laaja dokumentaatio luo ongelmia, kun ohjelmistoa korjataan tai muutetaan. Dokumentaatioon pitää tehdä samat korjaukset ja muutokset, mikä kasvattaa työtaakkaa huomattavasti. [4,7]

Ohjelmistotuotteet koostuvat pääasiassa komponenteista, jotka ovat toiminnallisia palasia. Eri ohjelmaversiot ovat monesti erilaisia monen komponentin rakennelmia, joita kutsutaan konfiguraatioiksi. Kun ohjelmasta on vuosien aikana tehty monta eri versiota, komponentteja voi olla satoja monessa erilaisessa kokoonpanossa. Tämä aiheuttaa yhteensopivuusongelmia. Jotta asiakkaan ohjelmassa ilmenneitä ongelmia voidaan lähteä ratkaisemaan, pitää tietää mikä versio ohjelmasta asiakkaalla on käytössä, koska sama ongelma ei todennäköisesti ole kuin parissa versiossa. Kunnollisen dokumentaation avulla saadaan selville kyseessä olevan ohjelmistoversion komponentit, sekä miten komponentit ovat toteutettu. [4]

Ilkka Haikala ja Jukka Märijärvi, kirjassaan Ohjelmistotuotanto [4], jakavat ohjelmistokehityksen dokumentaation kolmeen ryhmään:

- laatukäsikirja
- projektin hallinta
- tuotedokumentit.

Laatujärjestelmän dokumentit koostuvat laatukäsikirjasta, ohjeistuksista, dokumenttimalleista, laatujärjestelmän auditointiin liittyvistä raporteista ja tietyistä palaveripöytäkirjoista. [4]

Projektidokumentit koostuvat sopimuksesta, projektisuunnitelmasta, projektin seuranta raporteista ja loppuraportista. Projektidokumenteille ominaista on projekteihin liittyvä kertaluontoisuus ja ne arkistoidaan projektien päättyessä. [4]

Tuotedokumentit ovat tuotteen tai tuoteperheen dokumentaatio. Tuotedokumentit voidaan jakaa projektikohtaisiin ja tuotekohtaisiin tuotedokumentteihin. Organisaatio, joka tuottaa tuotteestaan jatkuvasti uusia versioita, tekee tuotedokumenteistaan sekä tuote- että projektikohtaiset versiot. [4]

Ian Sommerville jakaa julkaisussaan Software Documentation [7] ohjelmistodokumentaation kahteen ryhmään:

- Prosessidokumentaatio
- Tuotedokumentaatio

Prosessidokumentaatiossa käsitellään ohjelmiston kehitys- ja ylläpitoprosessia. Koska ohjelmisto itsessään ei ole konkreettinen laite, jonka toimintaa voi päätellä erilaisten toimilaitteiden yhteistyöstä, vaan tekstipohjaisista komponenteista koostuva abstrakti kokonaisuus, on koko ohjelmiston sekä ohjelmistokomponenttien toiminta dokumentoitava tarkasti. Prosessidokumentaatio jakautuu viiteen osa-alueeseen:

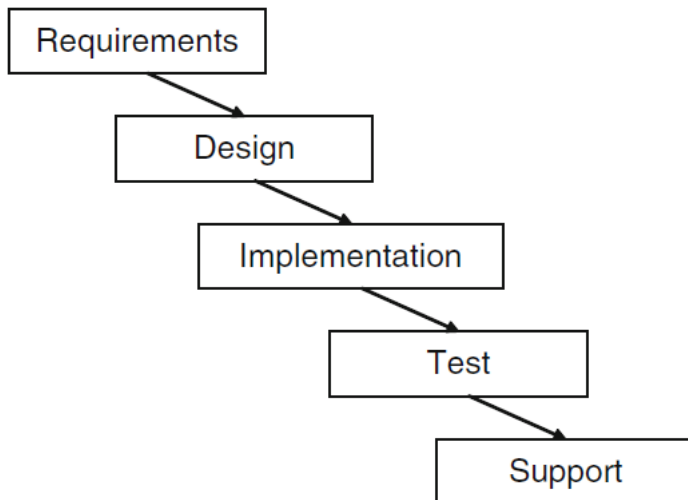
- Suunnitelmat, arvioinnit ja aikataulut
- Raportit
- Standardit
- Työraportit
- Memot ja viestiketjut

Suurin osa prosessidokumentaatiosta vanhentuu jo kehitysprosessin aikana, koska prosessin edetessä muutoksia ja uusia suunnitelmia joudutaan laatimaan. Osa vanhentuneistakin dokumenteista ovat hyödyllisiä. Esimerkiksi dokumentit, jotka kertovat suunnitteluvaiheen päätöksistä ja niiden perusteista, ovat myöhemmissä vaiheissa tärkeitä, jotta vältetään tekemästä ristiriitaisia päätöksiä. [7]

Tuotantodokumentaatio kuvailee kehitettävää ohjelmaa. Se voidaan jakaa kahteen dokumentaatiotyyppiin: Käyttäjä- ja järjestelmädokumentaatioon. Tuotanto dokumentaation tulee kehittyä samaa tahtia ohjelmiston kanssa, jotta se kuvaa ohjelmistoa tarkasti. [7]

Perus runko ja ajatus on aiemmin mainituissa lähteissä täsmälleen sama, jakaa dokumentaatio projekti/prosessi osaan sekä tuote-/tuotanto- osaan. Projekti/prosessi koostuu ohjelmiston tuotannon kannalta oleellisesta dokumentaatiosta sekä itse tuotannon aikana toteutetusta dokumentaatiosta. Tuote/tuotanto -osa on valmiin tuotteen kannalta oleellinen dokumentaatio. Selkeänä erona näiden kahden ohjelmistodokumentaatio mallin välillä on laatukäsikirja, mutta monissa organisaatioissa laadunvarmistusta tehdään muun projektin yhteydessä, jolloin laatuun liittyvän dokumentaation voi katsoa projekti/prosessi -dokumentaation osaksi.

Yleisesti ottaen perinteinen ohjelmistodokumentaatio on hyvin simppeliä, käytännönläheistä ja kattavaa, mutta hyvin jäykkää ja työlästä. Vesiputousmallissa jokainen vaihe suoritetaan loppuun, ennen seuraavaan vaiheeseen siirtymistä, eikä aikaisempiin vaiheisiin enää palata (Kuva 1). Mallin hyviä puolia ovat yksinkertaisuus toteuttamisessa ja projektin seuraamisessa, mutta vaatimuksia ja suunnitelmia ei voida muuttaa enää ohjelmointivaiheessa. Koska aiempiin vaiheisiin ei voi palata, valmiiseen tuotteeseen voi jäädä vikoja, jotka olisi voitu korjata muuttamalla ohjelmistolle suunniteltua rakennetta. Kesken projektin tulevat muutospyyntöt eivät sovellu vesiputousmallin toteutettaviksi ja muutokset pitää toteuttaa uutena päivitysprojektina. [8, 9]



*Kuva 1. Perinteinen vesiputousmalli [8]*

Teoriassa ohjelmistokehitys on pelkkää dokumentaatiota, koska ohjelmat luodaan tyypillisesti tekstipohjaisesti. Tällöin tuotantoprosessi on vaiheittainen kuvaus järjestelmän vaatimuksista niiden toteuttamiseksi. Jokaisessa prosessin vaiheessa luodaan spesifikaatiodokumentti seuraavan tason syötteeksi. Näiden spesifikaatiodokumenttien hyödyntämiseen ja esittämiseen tarvitaan kuvaustekniikoita. Ylimmän tason kuvaustekniikka on pääasiassa luonnollinen kieli ja alimmalla tasolla ohjelmiston toteutus esitettyä jollain toteutusvälineellä, esimerkiksi ohjelmointikielellä [4]

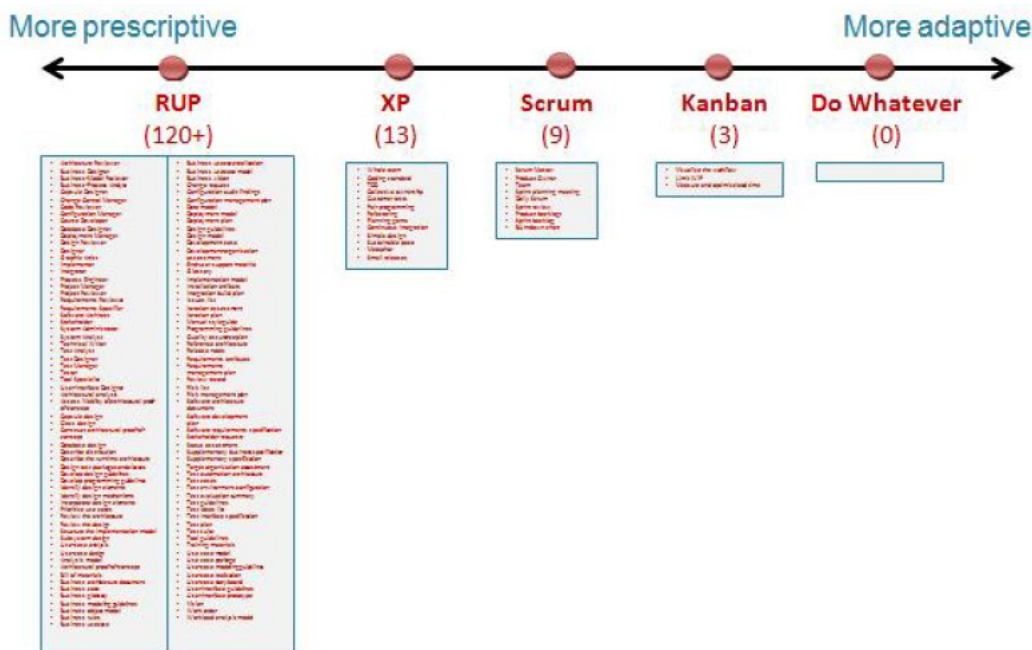
## 2.2 Ketterä ohjelmistokehitys

Ketterä (engl. agile) ohjelmistokehitys pyrkii vastaamaan tarpeeseen kevyestä ja ketterämmästä, ohjelmistokehitysprosessista. Ketterässä ohjelmistokehityksessä arvostetaan enemmän yksilöitä ja vuorovaikutusta kuin prosesseja ja työkaluja, enemmän toimivaa ohjelmaa kuin ymmärrettävää dokumentaatiota, enemmän yhteistyötä asiakkaiden kanssa kuin sopimusneuvottelua, enemmän muutokseen vastaamista kuin suunnitelman mukaan tekemistä. [5]

Toisin sanoen ketterässä ohjelmistokehityksessä dokumentaation sijaan keskitytään ohjelmakoodiin ja tieto kulkee verbaalisesti ohjelmistokehittäjien kesken. Käytännössä tämä tarkoittaa, että ohjelmistokehittäjien tulisi työskennellä yhteisessä tilassa, kehittäjien keskuudessa on asiantuntijoita, jotka ymmärtävät projektin ja pitävät huolta, että työskennellessä annetaan jatkuvasti palautetta, lyhyet inkrementointivälit, regressiotestit ovat täysin automatisoidut ja ohjelmisto kehittäjät ovat kokeneita. [5 s.13]

Ketterät menetelmät rakentuvat lyhyistä iteraatioista. Iteraatioiden lyhyiden ja testitapausten automaattisen suorittamisen ansiosta kehittämisen voi ajatella olevan jatkuvaa. Esimerkiksi ohjelmamuu-  
toksen tekeminen alkaa ohjelmoimalla muutokselle testitapauksia. Ajettaessa testitapaukset raportoivat testin epäonnistuneen, joten aloitetaan muutoksen ohjelmointi. Muutostöitä jatketaan, kunnes kaikki testitapaukset, sekä aikaisemmat kuin muutosta varten luodut saadaan virheettömästi ajettua läpi. [4]

Erilaisia ketteriä menetelmiä on monia erilaisia. Tämä johtuu siitä, että erilaiset yritykset tarvitsevat eritasoista vapautta. Joillakin aloilla myynnin ehtona on, että tehtävä tuote on tarkalleen kiveen hakattu ennen aloittamista, ja joissakin tuotteiden tulee vain täyttää tietyt asiakkaan vaatimukset. Kuvassa 2 esitellään erilaisten ketterien menetelmien vapautta työskennellä oman luovuuden mukaan. [6] Tässä diplomityössä keskitytään ketterien menetelmien osalta eXtreme programming (XP), Scrum, Kanban ja Lean menetelmiin.



**Kuva 2.** Ketterien menetelmien vapaus asteikko [6]

Martin Fowler, yksi Manifesto for Agile Software Developmentin [10] kirjoittajista, kertoo sivuillaan, että ketterien menetelmien yksi yleisistä ominaisuuksista on nostaa ohjelmointi keskeiseen rooliin koko ohjelmistokehityksessä. Tämä johtuu osiltaan siitä, että koodi määritellään suureksi osaksi dokumentaatiota tai jopa pääasialliseksi dokumentaatiotavaksi ohjelmistolle. [11]

Yleinen väärinymmärrys on, että koodi on ainoa dokumentaatio. Ajatukset siitä, että koodi on ainoa tapa dokumentoida, syntyy siitä, että koodia pidetään ainoana tarpeeksi yksityiskohtaisena ja tarkkana kuvaustapana ohjelmistosta. Heikkous koodin pitämisestä oleellisena osana dokumentaatiota on se, että kaikkien ohjelmoijien pitää tuottaa selkeää ja helposti luettavaa koodia. [11]

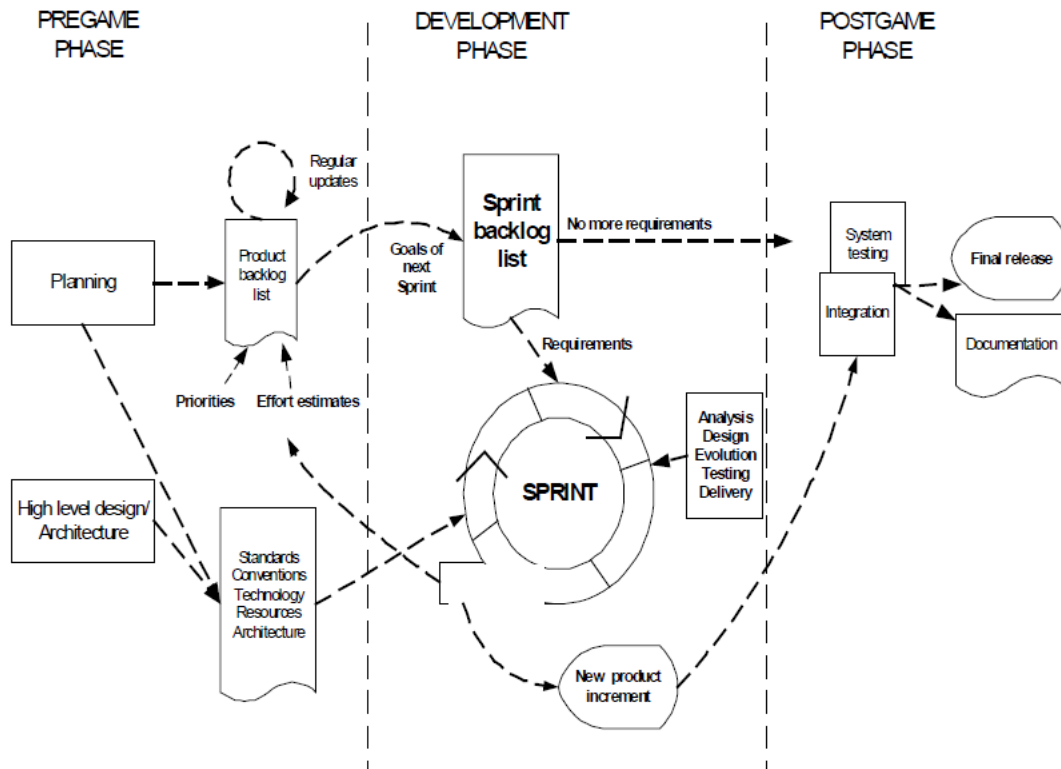
## 2.2.1 Scrum

Scrum nimitystä käytettiin ensimmäisenä Takeuchi ja Nonaka:n artikkelissa vuonna 1986. Scrum hyödyntää teollisuusprosessien kontrollointiteorioita järjestelmien kehittämiseen keskittyen joustavuuteen, mukautuvuuteen ja tuottavuuteen. Scrum ei määritä tiettyjä ohjelmistokehitystekniikoita implementointi vaiheeseen. Keskeisessä roolissa on ryhmän jäsenten toiminta, jotta järjestelmä olisi joustava alati muuttuvassa ympäristössä. [5]

Yleisellä tasolla Scrum:ssa organisaatio jaetaan pieniin ryhmiin, jotka tekevät työnsä itsenäisesti, mutta tarvittaessa toimivat myös sulavasti muiden ryhmien kanssa. Jokainen ryhmä jakaa heille annettun työn pienempiin tuotanto-osiin. Tuotanto-osien tärkeys koko projektissa ja sen toteuttamiseen kuluttava aika arvioidaan tarkasti. Tuotanto-osat jaetaan 1-4 viikkoisiin iteraatioihin. Jokaisessa iteraatiossa pyritään toteuttamaan yksi tuotanto-osa. Kun kaikki iteraatiot on suoritettu, pitäisi projektin tuote olla valmis. [6, 12]

Käytännössä Scrum ei ole näin yksinkertaista. Kuva 3 esittää Scrum-projektin kaavaa, jota työryhmä noudattaa. Scrum jakautuu kolmeen osa-alueeseen:

- Alkuvaihe (engl. Pregame phase)
- Tuotantovaihe (engl. Development phase)
- Jälkivaihe (engl. Postgame Phase).



**Kuva 3.** Scrum projektin etenemiskaava [5]

Alkuvaihe rakentuu suunnittelu- ja arkkitehtuuritason/korkean tason suunnitteluvaiheista. Suunnitteluvaiheessa määritellään kehitettävä järjestelmä listaamalla kaikki vaatimukset Product Backlog -listaan. Erilaisia vaatimuksia luovat asiakkaat, myynti ja markkinointiosasto, asiakastuki ja ohjelmistokehittäjät. Vaatimukset priorisoidaan sekä toteuttamisen työmäärä arvioidaan. Product Backlog:ia päivitetään uusilla ja yksityiskohtaisemmilla osioilla sekä tarkemmilla arvioilla ja uusilla prioriteettikäsityillä. Suunnittelu sisältää työryhmien, työkalujen ja muiden resurssien määrittelyn, riskiarvioinnit, ongelmien hallinta- ja koulutustarpeet sekä johdon hyväksynnän. Jokaisella iteraatiolla Scrum-ryhmä käy läpi päivitetyn Product Backlog:in, jotta tiedetään missä mennään ja mihin tähdätään (Kuva 3 sarake 1). [5, 12]

Arkkitehtuurivaiheessa suunnitellaan Product Backlog:in pohjalta järjestelmän arkkitehtuuri ja korkean tason malli. Olemassa olevaa järjestelmää paranneltaessa selvitetään, Backlog -kappaleiden implementointiin tarvittavat muutokset ja niiden luomat mahdolliset ongelmat. Suunnitelman katsauspalaverissa käydään läpi implementointiehdotuksia. Arkkitehtuurivaiheessa tehdään myös alustavat suunnitelmat julkaisun sisällöstä. [5]

Tuotantovaihe on Scrum:in ketterä osa. Tätä vaihetta käsitellään mustana laatikkona, jossa ennalta arvaamattomat tulokset ovat odotettavissa. Iteraatioiden aikana seurataan ja kontrolloidaan Scrum:ssa selvitettyjä erilaisia teknisiä ja ympäristö muuttujia. Tämän tyyppisiä muuttujia ovat mm. aikataulu,

laatu, vaatimukset, resurssit, käyttöönottotekniikat ja työkalut. Sen sijaan että nämä asiat olisi huomioitu vain ohjelmistokehitys projektin alussa, Scrum:ssa niitä pyritään kontrolloimaan koko ajan, jotta muutoksiin voidaan mukautua joustavasti (Kuva 3 sarake 2). [5]

Tuotantovaiheessa järjestelmää kehitetään iteraatioissa. Ne ovat kiertoja, joissa toiminnallisuutta kehitetään tai parannetaan tuottaakseen uusia lisäyksiä. Jokainen iteraatio sisältää ohjelmistokehityksen perinteiset vaiheet: Vaatimukset, analyysi, suunnittelu, kehitys ja toimitus vaiheet (Kuva. 3). Järjestelmän arkkitehtuuri ja malli kehittyvät iteraatioiden aikana. Yksittäinen iteraatio suunnitellaan kestämään viikosta kuukauteen. Yhdessä järjestelmäkehitysprosessissa voi olla kolmesta kahdeksaan iteraatiota ennenkö järjestelmä on valmis jakeluun. Iteraatioissa tehtävän komponentin tai ohjelmiston osan toteuttamiseen voi osallistua useampi työryhmä. [5]

Jälkipelivaihe sisältää julkaisun päättämisen. Tähän vaiheeseen siirrytään, kun on sovittu, että ympäristömuuttujat, kuten vaatimukset, on täytetty. Tässä tapauksessa ei enää löydy uusia kappaleita tai ongelmia eikä uusia voida keksiä. Järjestelmä on nyt valmis julkaistavaksi ja tähän valmistautuminen tehdään jälkipelivaiheessa. Tähän sisältyy muun muassa integraatio, järjestelmä testaus ja dokumentaatio (Kuva 3 sarake 3). [5]

## 2.2.2 EXTREME PROGRAMMING

Extreme programming:ssa (XP) käytetään yleisiä avainperiaatteita ja toimenpiteitä. Erityisen XP:sta tekee sen tapa yhdistää tutut toimenpiteet ja tavat tehokkaaksi ohjelmistokehityksen metodiksi. [5]

Useimmat lähteet kuvailevat XP:tä teoreettisin keinoin pureutumatta siihen, miten XP:tä toteutetaan käytännössä. Esimerkiksi Kent Beck ja Cynthia Andres kirjassaan Extreme Programming [13], Sharifah Syed-Abdullah, Mike Holcombe ja Marian Gheorghe tutkielmassaan Agile Methodology in Practice [14] ja Mike Holcombe Running an Agile Software Development Project [9], tyytyvät kuvaamaan XP:tä sen arvojen ja käytäntöjen pohjalta. Arvot ja käytännöt vaihtelevat hieman julkaisusta riippuen. Arvoja ovat kommunikaatio, yksinkertaisuus, palaute, rohkeus ja arvostus. Arvot ovat samoja kuin missä tahansa ketterässä menetelmässä. Pitää uskaltaa kysyä. Jos saman asian voi tehdä yksinkertaisestikin niin älä tee monimutkaista. Anna palautetta hyvästä ja pahasta, jotta tiedetään mihin suuntaan ollaan menossa. Vaikeuksien edessä rohkeasti eteenpäin. Arvostus ohjaa noudattamaan useita näistä ohjeista, jos arvostat työntekijöitä, niin annat paremmin palautetta, kommunikaatio on helpompaa ja rohkaisette toisianne. [9, 13, 14]

Käytäntöjä ovat suunnittelupeli, pari ohjelmointi, testaus ensin -ohjelmointi, yhteisomistus, jatkuva julkaiseminen ja ohjelmointitapa:

Suunnittelupelissä asiakas antaa asiakaskertomuksia (engl. customer stories), joiden pohjalta arvioidaan kuinka kauan ohjelman tuottaminen näiden kertomuksien pohjalta kestää. Näistä kertomuksista etsitään ohjelman kannalta olennaiset toiminnallisuudet, jotka kirjoitetaan pienille korteille. Korttien avulla voidaan luoda kuva toiminta sekvensseistä, joita ohjelmiston toteuttaminen vaatii. Näiden tietojen pohjalta luodaan testejä, joiden pohjalta testaus ensin ohjelmointi voidaan aloittaa. [9, 14]

Pari Ohjelmoinnissa kaksi työntekijää jakaa yhden tietokoneen ja kirjoittavat koodia yhdessä. Tällä tavoin työntekijät auttavat toisiaan keskittymään työhön. He voivat keskenään neuvotella erilaisista ratkaisuista. He ovat yhdessä vastuussa kirjoitetusta koodista. [9, 14]

Testaus ensin -ohjelmoinnissa kirjoitetaan epäonnistuva testi ennen kuin koodia muutetaan. Tällä tavoin käsitellään monta ongelmaa kerralla:

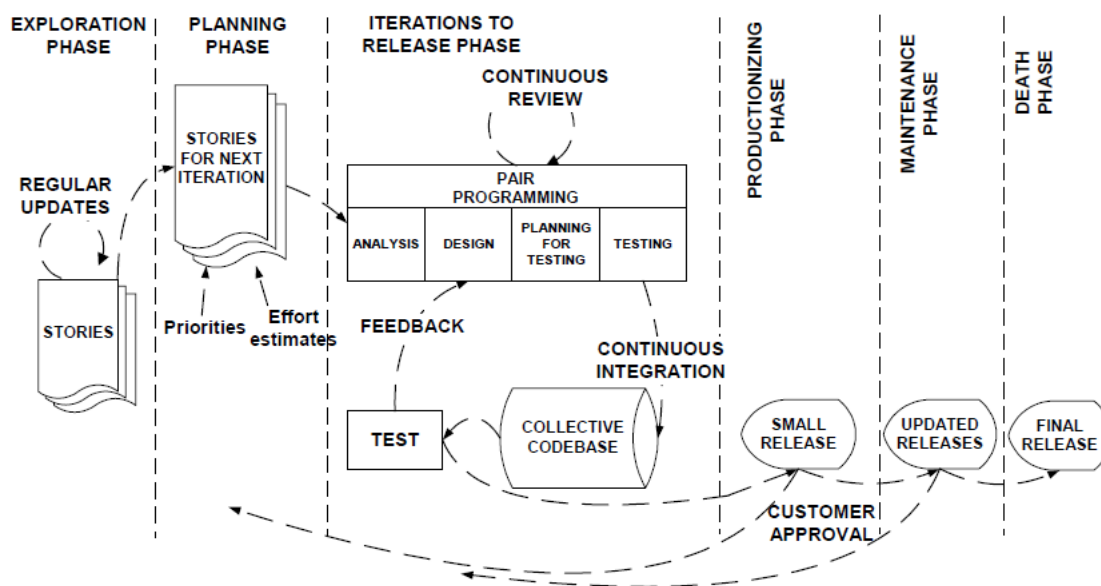
- Pidetään laajuus kurissa. Jos ohjelmoi huolettomasti, tulee ohjelmasta helposti suuri ja sekava. Jokainen ohjelmoitu tehtävä pitää olla tiettyä tavoitetta seuraava.
- Ohjelma on huonosti suunniteltu, jos testin kirjoittaminen on vaikeaa. Hyvin suunnitellusta ohjelmasta tietää mitä sen pitäisi tehdä ja mitä sen ei pitäisi tehdä.
- Ohjelmointirythmi: Testaa, kirjoita koodia, selvennä koodia (engl. refactor), testaa, kirjoita koodia, selvennä koodia. Tällä tavoin työ etenee loogisesti. [9, 14]

Yhteisomistus tarkoittaa, että kaikki ovat vastuussa ohjelmistosta ja sen ominaisuuksista. XP mallissa kommunikaatio on avointa ja yhteistyö on oletusarvo, minkä takia kaikilla on osansa muidenkin koodista. [9, 14]

Jatkuva julkaiseminen tarkoittaa sitä, että kun ohjelmisto täyttää sille asetetut päävaatimukset se otetaan käyttöön. Käyttäjät pääsevät käyttämään ohjelmistoa ja antamaan palautetta. Palautteen pohjalta ohjelmistoa parannellaan, kunnes se on riittävän hyvä. [9, 14]

Jokaisella on oma tapansa ohjelmoida, mutta yhteistyön mahdollistamiseksi yhteisiä ohjelmointitapoja tarvitaan. Esimerkiksi yhtenevä luokkien ja metodien nimeäminen. [9, 14]

P. Abrahamsson, O. Salo, J. Ronkainen ja J. Warsta kirjassaan A.G.I.L.E antavat yksityiskohtaisen kuvauksen siitä, miten XP toimii käytännössä. He jakavat Extreme programming:in elinkaaren kuuteen vaiheeseen: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death. [5]



Kuva 4. Extreme programming [5]

Exploration eli tutkintavaiheessa asiakas kirjoittaa asiakaskertomuksia, jotka pyritään sisällyttämään ensijulkaisussa. Asiakaskertomukset kuvaavat ohjelmaan lisättävät ominaisuudet. Tutkintavaiheessa projektia ajava ryhmä tutustuu projektin työkaluihin ja toimintatapoihin. Tuotantovaiheessa hyödynnettävät laitteet testataan ja prototyypin avulla tutkitaan mahdollisia tapoja järjestelmäarkkitehtuurin toteuttamiselle. Tutkintavaiheeseen käytetään yleensä parista viikosta pariin kuukauteen, riippuen siitä kuinka hyvin ohjelmistokehittäjät tuntevat järjestelmän (Kuva 4 sarake 1). [5]

Planning phase eli suunnitteluvaihe määrittää kertomuksien ja sovitun sisällön tärkeysjärjestyksen ensijulkaisua varten. Ensimmäiseksi arvioidaan, kuinka paljon työtä kertomuksien toteuttaminen vaatii. Arvion pohjalta sovitaan aikataulu. Ensijulkaisuun käytetty aika ei yleensä ylitä kahden kuukauden rajaa. Suunnitteluvaiheeseen käytetään pari päivää (Kuva 4 sarake 2). [5]

Iteraatiovaihe sisältää monta iteraatiota ennen kuin päästään ensijulkaisuun. Suunnitteluvaiheen aikataulu paloitellaan yhdestä viikosta neljän viikon mittaisiin iteraatiokierroksiin. Ensimmäisellä iteraatiolla luodaan koko järjestelmän arkkitehtuuri. Ensimmäiseen iteraatioon valitaan ne kertomukset, jotka vaativat koko järjestelmän rakenteen rakentamista. Asiakas päättää mitä kertomuksia iteraatioissa käytetään. Asiakkaan funktionaaliset testit ajetaan joka iteraation jälkeen. Viimeisen iteraation jälkeen järjestelmä on valmis tuotantoon (Kuva 4 sarake 3). [5]

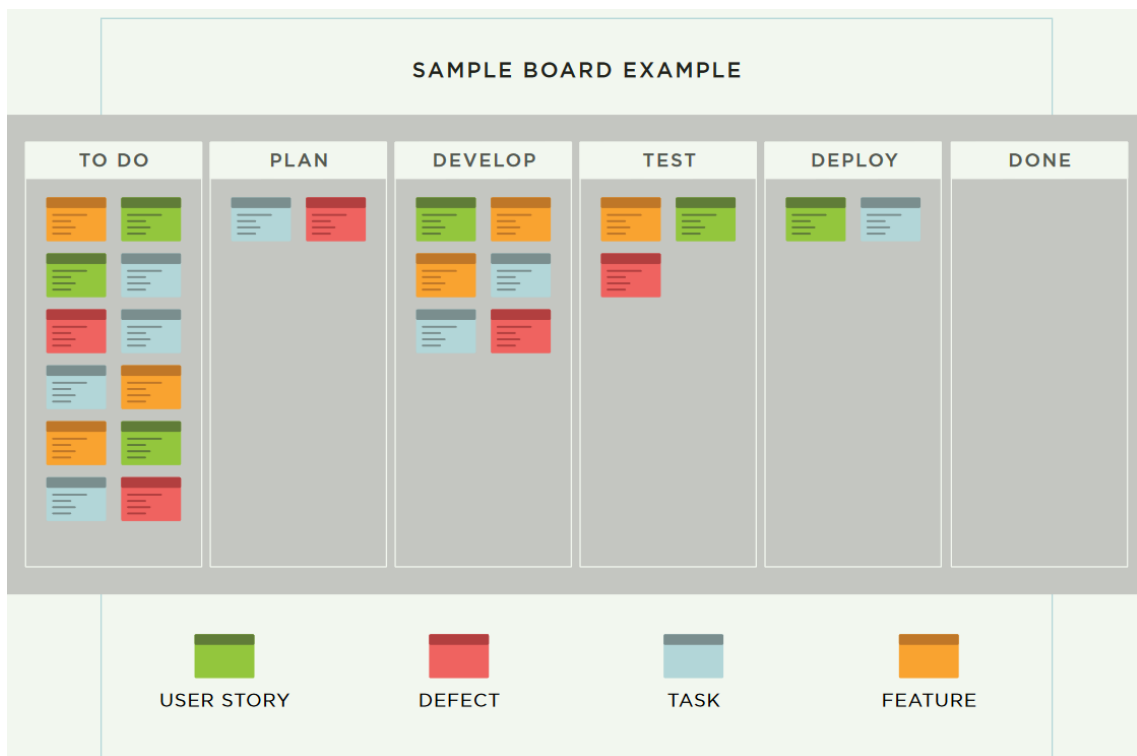
Productionizing phase eli tuotteistamisvaiheessa tehdään ylimääräisiä järjestelmän suorituskyvyn testauksia ja tarkastuksia, jotta järjestelmä voidaan julkaista asiakkaalle. Näissä testeissä voi löytyä uusia muutostarpeita ja päätös siitä toteutetaanko nämä muutokset tähän julkaisuun pitää tehdä nopeasti. Tässä vaiheessa voi olla tarvetta iteraatioiden nopeuttamiselle kolmesta viikosta viikkoon. Ideat ja ehdotukset dokumentoidaan myöhempää toteuttamista varten. Lykätty toteuttaminen voi sijoittua esimerkiksi ylläpitovaiheeseen (Kuva 4 sarake 4). [5]

Maintenance phase eli ylläpitovaihe käynnistyy siinä vaiheessa, kun ensijulkaisu on tuotteellisoitu ja XP projektin pitää ylläpitää tuotantoa ja tuottaa uusia iteraatioita. Ylläpitovaihe tarvitsee myös asiakastuen tehtäviä, minkä takia tuotanto nopeus voi hidastua, kun järjestelmä on tuotannossa. Monesti ylläpitovaiheessa pitää muuttaa ryhmän rakennetta ja kasvattaa ryhmäkokoja uusilla työntekijöillä (Kuva 4 sarake 5). [5]

Death phase eli kuolemavaihe lähestyy, kun asiakkaalla ei ole enää järjestelmään lisättäviä asiakaskertomuksia. Tässä tilanteessa asiakas on kaikin puolin tyytyväinen järjestelmään. Kuolemavaiheessa järjestelmästä kirjoitetaan vaaditut dokumentit, koska arkkitehtuuriin, suunnitteluun tai koodiin ei tule enää muutoksia. Kuolemavaihe voi ilmentyä myös tilanteissa, joissa järjestelmä ei tuota haluttuja tuloksia tai jos jatkokehitys kasvaa liian kalliiksi (Kuva 4 sarake 6). [5]

### 2.2.3 Kanban

Kanban:in ideana on tehdä työnteosta sulavampaa, tuoda esiin projektikulun ongelmakohdat, jotta niihin voidaan puuttua. Toyota kehitti Kanban toimintamallin 1940-luvulla, jotta varasto saataisiin vastaamaan kysyntää. Tässä toimintamallissa pyritään visualisoimaan työn kulkua käyttämällä tarralappuja kuvaamaan missä projektin vaiheessa tehtävät ovat (Kuva 5). Työkulun visualisoinnin avulla jo ensi silmäyksellä näkee, miten työ edistyy ja syntykö joissakin kohdissa ruuhkautumista. [15, 16]



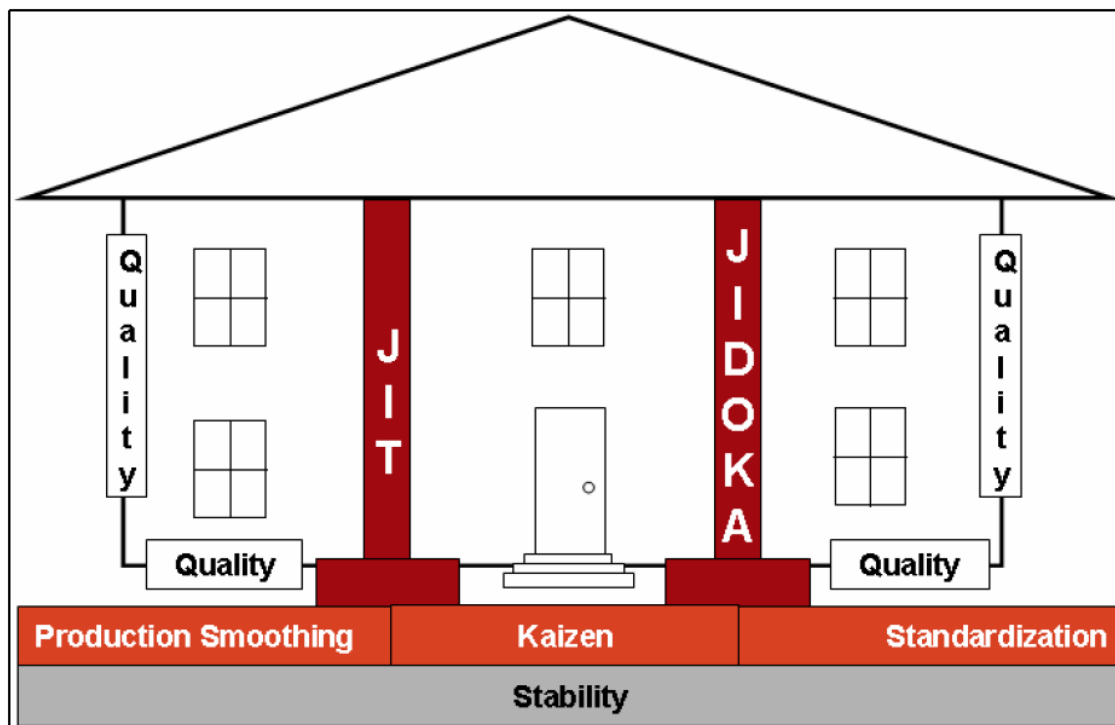
**Kuva 5.** Kanban [15]

Kanban:illa on kuusi perusperiaatetta: Visualisoi, rajoita keskeneräisen työn määrää, hallitse työnkulkua, tee täsmällisiä päätöksiä, palautteen keruu sekä parantaminen ja kehittyminen. [16]

Henrik Kniberg tuo esille kirjassaan Kanban vs Scrum, että nämä ovat vain työkaluja ja sen takia ei tarvitse sitoutua vain yhteen. Näistä työntekoa helpottavista työkaluista pitää ottaa ne osat käyttöön, jotka omaa työtä parantavat ja helpottavat, ja jättää pois ne, jotka koetaan huonoiksi. Kirjassaan hän esittää esimerkin huollon ja kunnossapidon näkökulmasta, jossa työt eivät ole aina ennalta nähtävissä ja tästä syystä on vaikea sitoutua mihinkään tiettyyn iteraatio pituuteen. Henrik ehdottaakin Scrum:in ja Kanban:in tekniikoiden yhdistämistä. [6]

## 2.2.4 Lean

Vuonna 1949 Toyotan myyntiluvut lähtivät kovaan laskuun, minkä seurauksena Toyota kehitti oman tuotantojärjestelmän Toyota Production System (TPS). 1990 Womack, Jones ja Ross esittelivät Leanin, sekä TPS, että Lean etsivät keinoja vähentää läpimeno-aikaa poistamalla kaikki ylimääräinen. Termit ”Lean” ja ”TPS” ovat synonyymejä. [17]

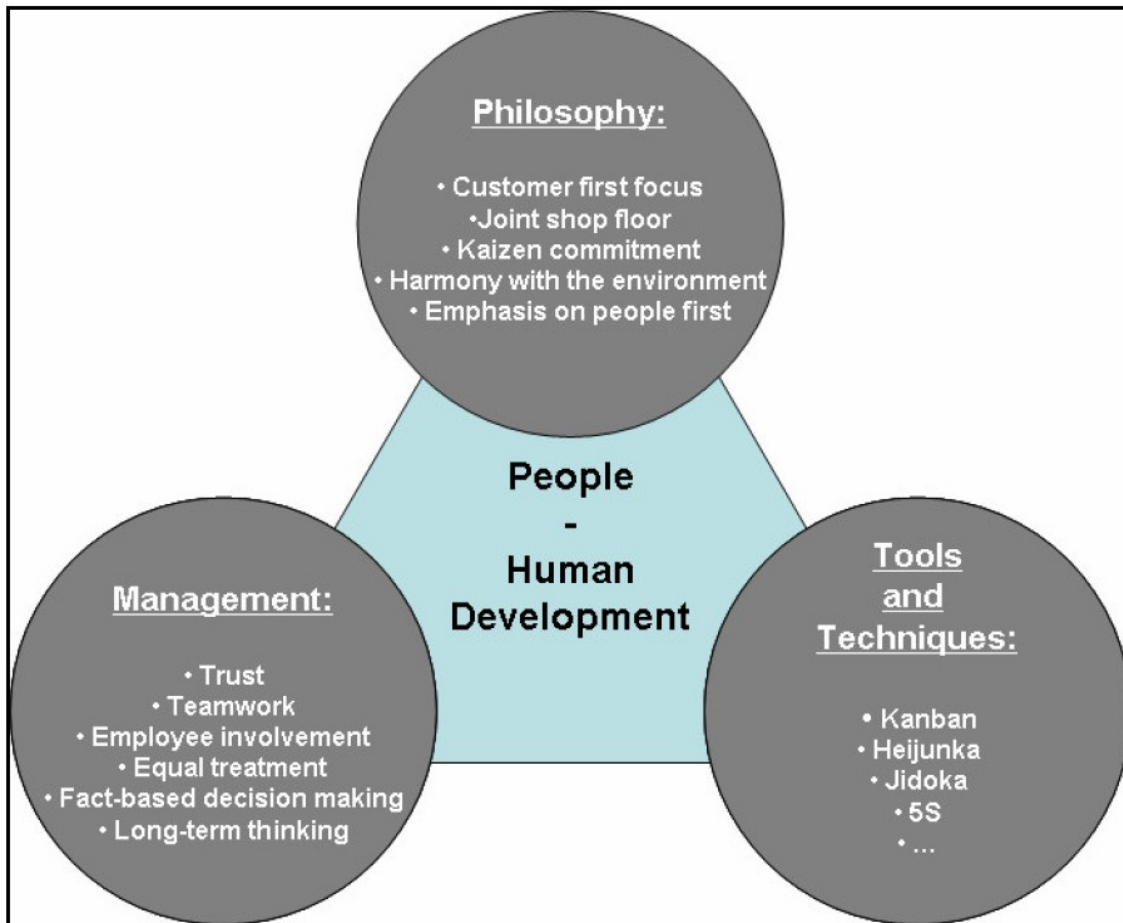


*Kuva 6. Toyota Production System tukipilarit [17]*

JIT eli just-in-time pyrkii prosessi virtaan, jossa oikea määrä oikeita kappaleita tulee tuotantolinjalle juuri tarvittavaan aikaan. Tavoitteena on tuotanto ilman keskeneräisvarastoa (Kuva 6). [17]

Jidoka on automaatiota ihmisen kosketuksella. Tarkoituksena ovat tuotantolaitteet, jotka pysähtyvät automaattisesti, jos laite tuottaa viallisen tuotteen. Tämä estää useamman viallisen tuotteen luomista ja vähentää hukkaa huomattavasti. Automatisointi vähentää työvoiman tarvetta ja mahdollistaa yhden työntekijän ajavan useaa tuotantolaitetta (Kuva 6). [17]

Kaizen tarkoittaa jatkuvaa parantamista. Se on yritysfilosofia, jossa pyritään parantamaan yrityksen prosesseja pienillä, mutta tehokkailla tavoilla. Toyota kehitti Kaizen:in 1950-luvulla ja sen päälle kehitettiin TPS (Kuva 6). [11, 18]



*Kuva 7. Toyota tuotantojärjestelmä kolmio [17]*

Gary Convisin loi TPS kolmion (Kuva 7), koska hänestä tuntui, että Ohnon TPS teoriota ymmärrettiin väärin, kun johtajat yrittivät hyödyntää vain yksittäisiä osa-alueita TPS:stä. Hänen mukaan onnistuneeseen TPS:n käyttöön tarvitaan kaikkien sitoutuminen. [17]

Monesti luullaan, että Agile ja Lean ovat vain eri nimet samalle asialle, mutta kirjallisuudessa näitä selvästi pidetään eri asioina. Erään näkemyksen mukaan Lean on joukko periaatteita, kun taas Agile on käytännön toimintaa. Toisin sanoen Lean on näkökulma, jolta toimintaa katsotaan ja Agile on tapa tehdä asioita. Tämän näkemyksen pohjalta on aloitettu tutkimuksia, voiko Agile:n ja Lean:in yhdistää. [19]

Esimerkiksi Agilessä keskitytään tiiviiseen yhteistyöhön asiakkaan kanssa ja tuotteen nopeaan toimitamiseen, kun taas Lean:issa keskitytään poistamaan ylimääräiset asiakkaan toiveista. [19]

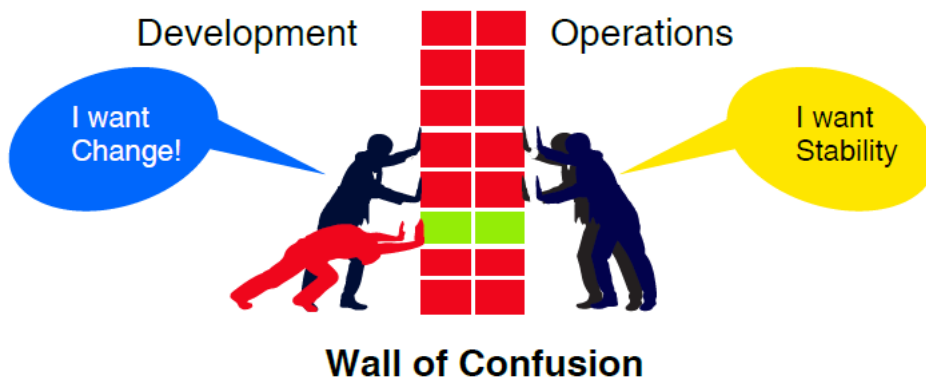
Näiden erojen pohjalta Lean:lle on syntynyt kaksi erilaista toteutustapaa. Ensimmäisessä tavassa Lean toteutetaan ylhäältä alas, jolloin Agilen alhaalta ylöspäin toteutus toimii menestyksekkäästi. Yrityksen pyrkiessä Agile -malliin täytyy suurten muutosten tulla yrityksen huipulta. Organisaatiostrategia toimii kontekstina, jossa Agile -prosessit toimivat tehokkaasti. [19]

Toisessa tavassa Lean auttaa Agilen kasvattamisessa koska Leania voi hyödyntää alueilla joihin Agile ei sovellu. Suurissa yrityksissä on havaittu, että suositut Agile metodit, kuten Scrum, ei kasva sisältämään ohjelmointi-, tuotanto- ja organisaatiotasoa ilman muutoksia. Tarvittavien muutosten perustana toimii Leanin periaatteet muri, mura and moda joiden on todettu toimivan suuren mittakaavan Agile -projekteissa. [19, 20]

## 2.2.5 DevOps

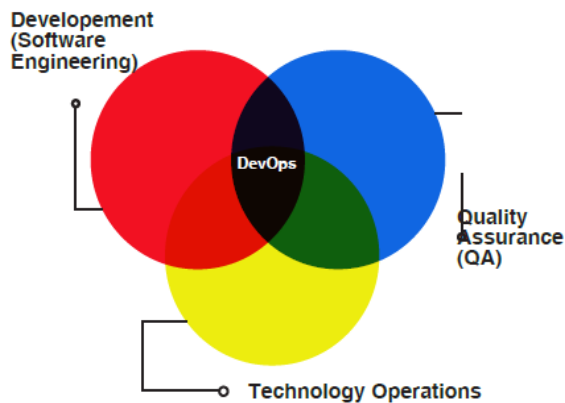
Uusien ohjelmistoteknologioiden, kuten pilvipalveluiden ja puhelin applikaatioiden seurauksena ohjelmistokehitystyön tarpeet ovat muuttuneet. Ennen tehtiin yksittäinen ohjelma, joka valmistuttuaan laitettiin myyntiin ja tässä vaiheessa kehitystiimin työt tämän projektin osalta päättyivät. Nykymarkkinoilla monet ohjelmistot vaativat ohjelmiston valmistumisen jälkeen ylläpitoa, päivittämistä ja seuranta. Perinteisesti ohjelmistoyrityksessä on kolme ydinryhmää: tuotanto, testaus ja toimitus. Tuotannon (engl. development) vastuulla on uusien ohjelmien, verkkosivujen tai tietopankkien luominen ja toimituksen (engl. operations) vastuulla on niiden ylläpito. [21, 22]

Karthiga Sadasivan ja James Lee ottavat kirjoissaan DevOps ja A Pragmatic Guide to Getting Started with DevOps esille, kuinka tuotanto- ja toimitusryhmien välille syntyy epäselvyyden seinä (engl. wall of confusion) (Kuva 8). Seinä syntyy, kun saman projektin eri ryhmillä on vastakkaiset tarpeet. Tuotantoryhmä pyrkii aina luomaan uutta ja erilaista, kun taas toimitusryhmä haluaa pitää järjestelmän vakaana, ja muutokset toimivaan järjestelmään luovat epävakautta. Perinteisessä järjestelmässä tuotantoryhmä tekee ohjelman ja antaa sen testaukseen. Tuotanto ja testaus tekevät yhteistyötä, kunnes ohjelmisto on valmis, jolloin ohjelmisto annetaan toimitusryhmälle. Tässä vaiheessa törmätään käyttöönotto-ongelmiin. [23, 24]



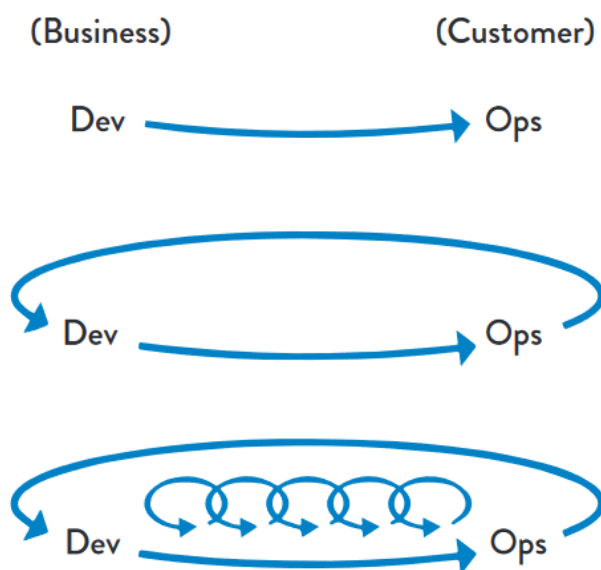
*Kuva 8. Epäselvyyden seinä [24]*

Patrick Debois kehitti termin DevOps kuvaamaan ohjelmistokehitystä, joka tähtää kehityksen (engl. development) ja toimituksen (engl. operations) saumattomaan yhteistyöhön. DevOps:ssa tuotanto, testaus ja toimitus toimivat yhtenä isona ryhmänä, jolloin käyttöönotto-ongelmat havaitaan, testataan ja korjataan prosessin aikana eikä käyttöönoton jälkeen tai sen yhteydessä (Kuva 9). [22, 23, 24]



*Kuva 9. DevOps tuotannon, testauksen ja toimituksen osalta [24]*

Vain harva julkaisu ottaa kantaa DevOps:in perusluonnetta pidemmälle ja esittelee peruseriaatteita, joiden mukaan DevOps toimii. Useimmat julkaisut viittaavat Gene Kim:in, Kevin Behr:in ja George Spafford:in kirjoissa DevOps Handbook ja The Phoenix Project: A Novel About IT esiteltäviin kolmeen tapaan: Virtaus (engl. flow), palaute (engl. feedback) sekä jatkuva oppiminen ja kokeilu (engl. continuous learning and experimentation). [25, 26]



**Kuva 10.** DevOpsin kolme tapaa. Ylin virtaus, keskimäinen palaute, alin oppiminen ja kokeilu [25]

Ensimmäinen tapa (Kuva 10, ylimmäinen) keskittyy työvirran tehostamiseen. Työ pitää tehdä näkyväksi esimerkiksi Kanban:in avulla. Tuotantokokoa ja yksittäisten työtehtävien kokoa pienennetään. [25, 26]

Muutamia ensimmäisen tavan toimintatapoja [25, 26]:

- Jatkuva integraatio (engl. Continuous integration)
- Jatkuva toimitus (engl. Continuous delivery)
- Jatkuva käyttöönotto (engl. Continuous deployment)
- Arvovirran kartoitus (engl. Value stream mapping)
- Kanban

Toinen tapa (Kuva 10, keskimäinen) keskittyy palautteen hyödyntämiseen. Palautteen avulla pyritään estämään ongelmien toistumista sekä tehostamaan ongelmien havaitsemista ja ratkaisua. Tämä kasvattaa tuotteiden laatua ja työntekijöiden tietotaitoa. [25, 26]

Muutamia toisen tavan toimintatapoja [25, 26]:

- Automaattinen testaus (engl. Automated testing)
- Tuotantomuutosten vertaisarviointi (engl. Peer review of production changes)
- Seuranta/Tapahtuma tiedon hallinta (engl. Monitoring/Event management data)
- Tuotannon lokikirjat (engl. Production logs)
- Prosessimittaukset (engl. Process measurements)
- Muutosten, tilanteiden, ongelmien ja tiedon hallinta (engl. Change, incident, problem and knowledge management data)

Kolmas tapa (Kuva 10, alimmainen) keskittyy jatkuvaan toimintaan. Jatkuva toiminta lähtee työntekijöistä, minkä vuoksi pitää luoda jatkuvan oppimisen ja kokeilemisen työskulttuuri. Tällöin yksittäiset työntekijät saavat mahdollisuuden kartuttaa omaa tietotaitoaan, joka lopulta leviää ryhmän ja koko organisaation tietotaidoksi. Tärkeitä osia oppimisen ja kokeilemisen työskulttuurissa ovat luottamus, mahdollisuus riskinottoon ja tieto siitä, että yrityksessä keskitytään enemmän virheiden korjaamiseen, kuin syyllisten etsintään. [25, 26]

Muutamia kolmannen tavan toimintatapoja [25, 26]:

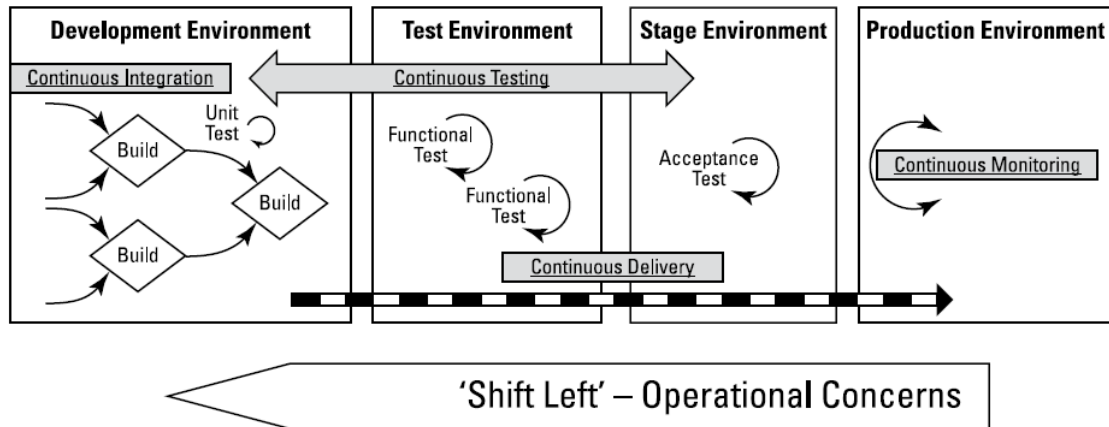
- Oppiminen ja kokeileminen
- Deming'in kierto. Suunnittele, tee, tarkista ja toimi
- Parantamisen kata. Lean'in perusajatus kuinka pyrkiä aina paremmaksi

Sanjeev Sharma ja Bernie Coyne esittelevät kirjassaan DevOps for dummies neljä peruseriaatetta [21]:

- Tuota ja testaa tuotannon tyyppisiä järjestelmiä vastaan (engl. Develop and test against production-like systems)
- Käyttöönnotta luotettavien ja uudelleen käytettävien prosessien avulla (engl. Deploy with repeatable, reliable processes)
- Seuraa ja validoi toimituksen laatua (engl. Monitor and validate operational quality)
- Voimista palaute kiertoja (engl. Amplify feedback loops)

DevOps konseptiin sisältyy vasemmalle siirtyminen (engl. shift left, Kuva 11), jossa toimituksen työpanosta hyödynnetään jo tuotannossa. Tuotannon ja toimituksen työteko helpottuu ja työnlaatu parantuu, kun kehitys ja testaus tehdään tuotannontyyppisiä järjestelmiä vastaan. Tällöin nähdään heti, kuinka sovellus käyttäytyy ja kuinka hyvin sovellus toimii, eikä vasta toimitusvaiheessa. [21]

Jos sovellusta päästään testaamaan tuotannontyyppisessä järjestelmässä jo tuotantokierroksen alussa, saadaan kaksi merkittävää etua. Tällöin sovellusta voidaan testata ympäristössä, joka vastaa toimintaympäristöä, johon se lopulta asennetaan. Toisena etuna voidaan heti testata ja validoida sovelluksen toimitusprosessit. [21]



*Kuva 11. DevOps:in perusajatus vasemmalle siirtymisestä [21]*

Käyttöönnotossa luotettavien ja uudelleen käytettävien prosessien avulla pyritään siihen, että toimitus tukisi ketteriä, tai edes iteroivia ohjelmistokehitysprosesseja tuotannosta asti. Usein tehtäviä, toistettavia, iteroivia ja luotettavia prosesseja luodaan automaation avulla, minkä vuoksi organisaation täytyy luoda tuotantolinja, joka mahdollistaa jatkuvan, automaattisen käyttöönoton ja testauksen. [21]

Toistuva käyttöönotto mahdollistaa käyttöönottoprosessien testauksen, minkä avulla varsinaisessa käyttöönotossa esiintyvien vikojen riski saadaan alhaiseksi. [21]

Yleisellä tasolla yritykset ovat hyviä sovellusten ja tuotettavien järjestelmien seurannassa, koska tuotantojärjestelmien mittauksia voidaan kerätä reaaliajassa. Usein yritysten seuranta on erillistä ja yksittäistä. 'Seuraa ja validoi toimituksen laatua' -periaate siirtää seuranta aiemmaksi, mutta sitä varten automatisoitu testaus pitää myös aloittaa aiemmin. Jatkuva ja aikaisin aloitettu automatisoitu testaus mahdollistaa sovelluksen toiminnallisten (engl. functional) sekä passiivisten (engl. non-functional) ominaisuuksien seurannan. Toistuvalla seurannalla saadaan ennakkovaroitus toimitus ja laatuongelmista, joita voi syntyä tuotannosta. [21]

DevOps pyrkii mahdollistamaan yrityksiä reagoimaan ja tekemään muutoksia nopeasti. Ohjelmistotuotannossa siihen vaaditaan nopeaa palautetta ja nopeaa oppimista jokaisesta toiminnosta. Yrityksen pitää luoda kommunikaatiokanavia, jotka mahdollistavat sidosryhmän käsitellä ja toimia palautteen mukaan. Palautteesta riippuen esimerkiksi tuotanto voi parantaa tuotantoympäristöä. [21]

### 2.3 Rautatiestandardien vaatimukset dokumentaatioon

Tässä kappaleessa käsitellään rautatiestandardeja IEEE 1558 ja EN 50128 sekä niiden asettamia vaatimuksia eri tasojen ohjelmistodokumentaatioon.

### 2.3.1 IEEE 1558

IEEE 1558 standardi käsittelee ohjelmistodokumentaatiota rautateillä käytettävistä laitteista ja järjestelmistä. Standardi luotiin, koska alalle alkoi tulla eriäviä vaateita ohjelmistodokumentaatiolle ja tarvittiin standardi yhtenäistämään tarpeet. Sen avulla alan ammattilaiset pystyvät luomaan tasalaatuista dokumentaatiota, joka helpottaa ohjelmiston toiminnallisuuden ja päivittämisen ymmärtämistä sekä yritysten välistä yhteistyötä. [3]

Standardissa esitellään dokumentit, kerrotaan lyhyesti mitä niiden tulisi sisältää, minkälainen rakenne dokumenteilta vaaditaan sekä listaa dokumentaatiotyypit [Taulukko 1]. Asiakas päättää dokumentaatiotyypin omaa tarvettaan ajatellen. Dokumentaatiotyypin valinta vaikuttaa asiakkaan osallisuuteen riskeistä ja riskien hallinnasta. [3]

Ykköstyypin dokumentaatio (Type 1) käsittelee vain ohjelmiston asennusta ja käyttöä [3]

Kakkostyypin dokumentti käsittelee ohjelmiston asentamisen ja käytön lisäksi ohjelmistoprojektin hoitamista. [3]

Kolmostyypin dokumentti sisältää aikaisemmat ja ohjelmiston toiminnalliset vaatimukset/rajapinnat sekä ohjelmistotestauksen tulokset. [3]

Nelostyypin dokumentissa on kaikki ohjelmistoprojektin hallinnan asiat, ohjelmiston toiminnalliset vaatimukset/rajapinnat, ohjelmistotestauksen suunnittelu ja toteutus sekä tulokset, vaatimusten verifiointi ja validointi sekä ohjelmiston asentaminen ja käyttö. [3]

Vitostyypin dokumentti sisältää nelostyypin asiat sekä yksityiskohtaisen ohjelmistosuunnittelun. [3]

Standardi määrittelee mitä dokumentteja ohjelmistokehityksessä tulisi luoda ja toimittaa asiakkaalle. [3]

**Taulukko 1: Dokumenttien sisältö tyypit [3]**

**Table 1—Deliverables**

Document	Abbreviation	Type 1	Type 2	Type 3	Type 4	Type 5
Software project management plan	SPMP		x	x	x	x
Software quality assurance plan	SQAP				x	x
Software configuration management plan	SCMP				x	x
Software verification and validation plan	SVVP				x	x
Software verification and validation report	SVVR				x	x
Software requirements specification	SRS			x	x	x
Interface control document	ICD			x	x	x
Software design description	SDD					x
Database design description	DBDD					x
Software requirements traceability matrix	SRTM			x	x	x
Software test plan	STP				x	x
Software test procedure	STPr				x	x
Software test report	STR			x	x	x
Software version description	SVD	x	x	x	x	x
Software user manual	SUM	x	x	x	x	x

Dokumenttityyppien ja niiden sisällön lisäksi IEEE 1558 sisältää oman kappaleensa dokumenteilta vaaditulle rakenteelle. IEEE 1558 vaatii, että dokumentin jokaisella sivulla näkyy dokumentin nimi, dokumentin numero, dokumentin tarkastustaso, sivunumero sekä sivujen kokonaismäärä. Dokumentin runko jaetaan pää- ja alaotsikoihin. Alaotsikoita saa käyttää vain, jos niitä on enemmän kuin yksi. Otsikointi käyttää normaalia numerointia: 1.0 1.1 1.1.1 1.1.2 1.2 1.3 [3]

Dokumentin tulee sisältää kansilehden, tarkastuslehden, sisällysluettelon, kuvaluettelon, taulukkoluet-  
telon sekä liitteet. Otsikkosivu sisältää projektin nimen, dokumentin nimen lyhenteineen, järjestelmän  
nimen, tuottajan nimen, tarvittaessa CDRL tunnusteen, osallisten nimet, dokumentin numeron, tarkas-  
tustason ja päiväyksen, dokumentin tilan ja tarvittaessa salassapitoehdot. [3]

Osallisiin kuuluu henkilöitä, jotka ovat vastuussa dokumentin valmistelusta, läpikäymisestä ja hyväk-  
symisestä. Kansilehdellä tulee olla jokaisen nimi, titteli ja allekirjoitus. [3]

Tarkastuslehti sisältää taulukon, johon on merkitty tarkastusnumero, ajankohta ja kuvaus muutoksista.  
Kuvaus muutoksista sisältää viittaukset kohtiin, joita muutettiin. Varsinaista muutoshistoriaa käsitel-  
lään erillisessä dokumentissa. Jokaisessa muutokerrassa pitää olla niiden henkilöiden nimet, jotka  
valmistelivat, kävivät läpi ja hyväksyivät julkaisun. [3]

Sisällysluettelon pitää sisältää kaikki pääotsikot ja sivunumerot. Jos pääotsikon alla on useita alaotsi-  
koita, niiden pitää näkyä sisällysluettelossa sivunumeroineen. Alaotsikoita voi lisätä sisällysluette-  
loon, mikäli lisäykset auttavat tiedon löytämisessä. Liite osio kuuluu sisällysluetteloon. [3]

Jos dokumentti sisältää enemmän kuin yhden kuvan, pitää dokumentissa olla kuvaluettelo, johon kirjataan tunniste, kuvaus ja sivunumero. [3]

Jos dokumentissa on enemmän kuin yksi taulukko, pitää dokumentissa olla taulukkoluetelo, johon kirjataan tunniste, kuvaus ja sivunumero. [3]

### 2.3.2 EN 50128

EN 50128 on ohjelmistostandardi rautateiden ohjaus- ja turvajärjestelmille. Rautatieympäristössä oleville ohjelmistoille on oma standardiryhmänsä, johon kuuluu EN 50128 lisäksi myös EN 50126 ”Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)” ja EN 50129 ”Railway applications – Safety related electronic systems for signalling”. Rautatiestandardit käyttävät turvallisuuden eheys tasoja SIL (Safety Integrity Level) ja EN 50128 Standardissa käydään läpi minkälaisia metodeja pitää käyttää, jotta ohjelmisto vastaa Standardien vaatimia SIL tasoja. [2]

SIL tasojen avulla validoidaan järjestelmän turvallisuuden tasoa. Vaatimukset kasvavat mitä vaarallisemman seurauksen ohjelmistovirhe aiheuttaa. Korkeamman turvallisuustason saavuttaminen vaatii huolellisempaa ja laajempaa dokumentaatiota. Tasoja on viisi, jotka alkavat nollassa ja päättyvät tasoon neljä (Kuva 12). Nollassa järjestelmä luokitellaan ei turvallisuuteen liittyväksi ja tason neljä järjestelmä luokitellaan korkean turvallisuuden järjestelmäksi. Tason nolla ylittävissä järjestelmissä virhe voi aiheuttaa kuoleman, mutta korkeampi SIL taso voidaan asettaa myös rahallisten, materiaali- ja ympäristöhaittojen takia. [1, 2]

Software safety integrity level	Description of software safety integrity
4	Very High
3	High
2	Medium
1	Low
0	Non safety-related

**Kuva 12.** SIL tasot ja tasojen kuvaukset [2]

SIL0 tason saavuttamiseksi EN50128 standardin vaatimukset täyttäen suositellaan, että projektista on olemassa ohjelmiston suunnitteludokumentit (engl. software planning documents), ohjelmistovaatimusten dokumentit (engl. software requirements documents), lähdekoodi ja sen dokumentaatio (engl. source code & documentation), ohjelmiston validointiraportti (engl. software validation report) ja ohjelmiston ylläpitorekisterit (engl. software maintenance records) [Taulukko 2]. [2]

**Taulukko 2:** Dokumentaation SIL suositukset ohjelmistokehityksen elinkaareissa [2]

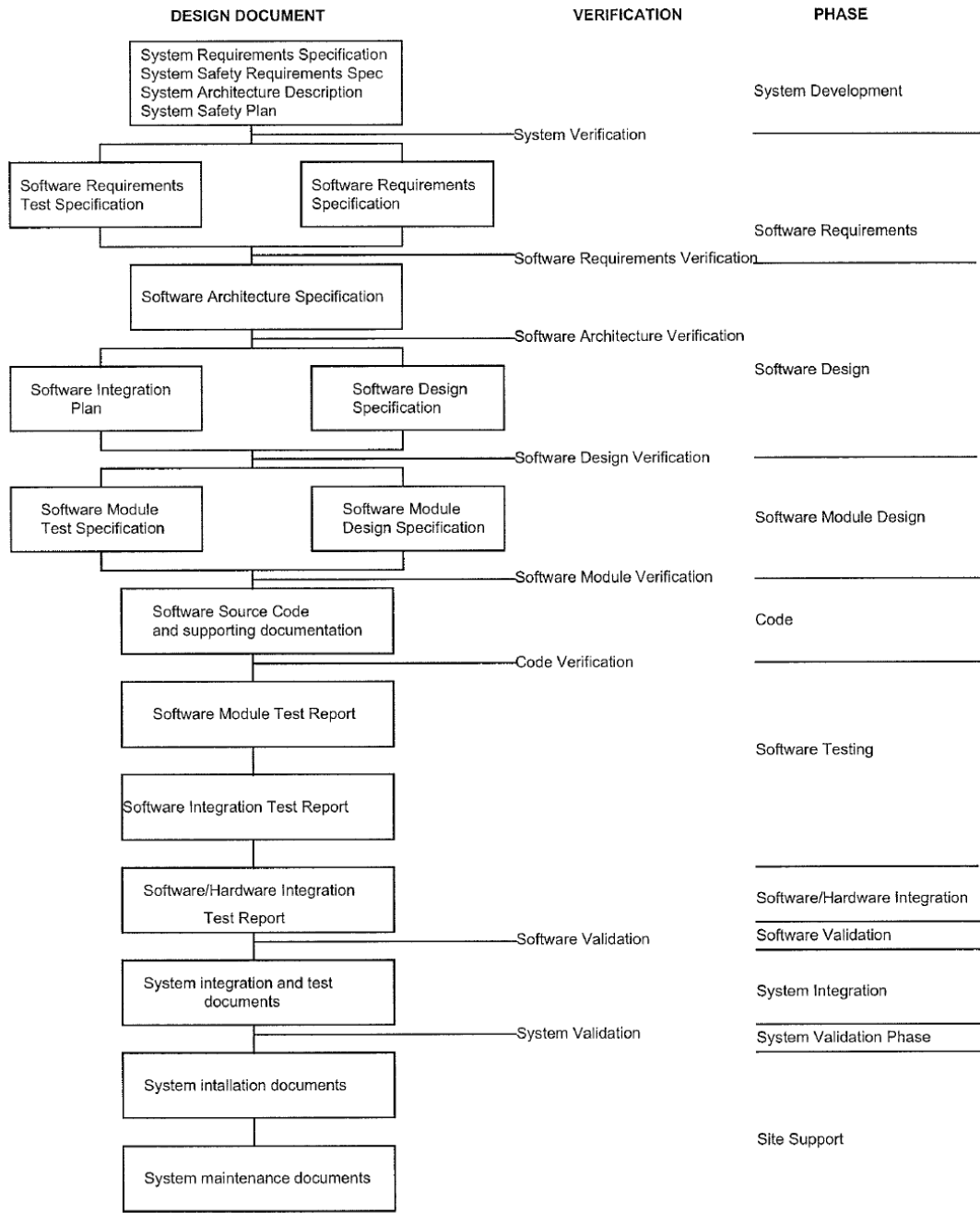
DOCUMENTATION	SWS IL 0	SWS IL1	SWS IL2	SWS IL3	SWS IL4
1. Software Planning Documents	R	HR	HR	HR	HR
2. S/W Requirements Documents	R	HR	HR	HR	HR
3. S/W Design Documents	-	HR	HR	HR	HR
4. S/W Module Documents	-	HR	HR	HR	HR
5. Source Code & Documentation	R	HR	HR	HR	HR
6. S/W Test Reports	-	HR	HR	HR	HR
7. S/W & H/W Integration Test Report	-	HR	HR	HR	HR
8. S/W Validation Report	R	HR	HR	HR	HR
9. S/W Assessment Report	-	HR	HR	HR	HR
10. S/W Maintenance Records	R	HR	HR	HR	HR
Requirement  Compliance with EN ISO 9000-3 implies the production of adequate documentation for all Software Safety Integrity Levels. For Software Safety Integrity Level 0, the designer shall choose suitable types of document.					

Osalle dokumenteista on standardissa täsmennetty, kenen vastuulle niiden luominen kuuluu.

**Taulukko 3:** EN 50128 Vastuut [2]

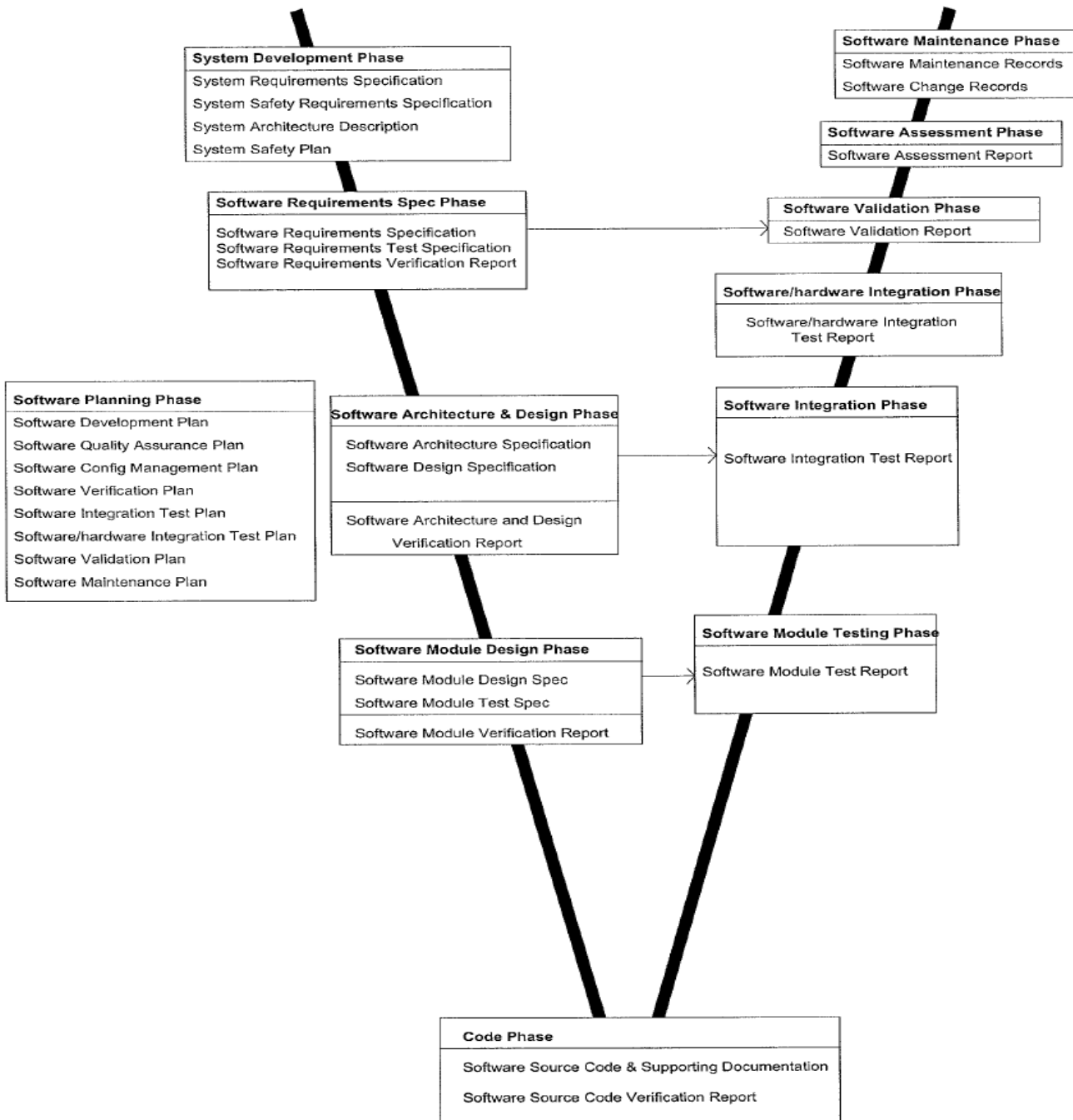
Software Requirements Specification	Designer
Software Requirements Test Specification	Validator
Software Architecture	Designer
Software Design and Development	Designer
Software Verification and Testing	Verifier
Software/Hardware Integration	Designer
Software Validation	Validator
Software Assessment	Assessor

EN50128 standardi esittelee ohjelmistokehityksen elinkaarta sekvenssikaaviolla ja v-mallilla (Kuva 13,14). Malleista huomataan, että standardi käsittelee ohjelmistokehitystä vain perinteisenä lineaarisena projektina, jossa suunnittelu tehdään ensin, sitten toteutetaan koodi ja sitten testataan. Nämä mallit eivät sovellu ketterään ohjelmistokehitykseen, jossa iteraatioissa tehdään koodausta, testausta sekä suunnittelua.



**Kuva 13.** EN50128 Ohjelmistokehityksen elinkaari versio 1 [2]

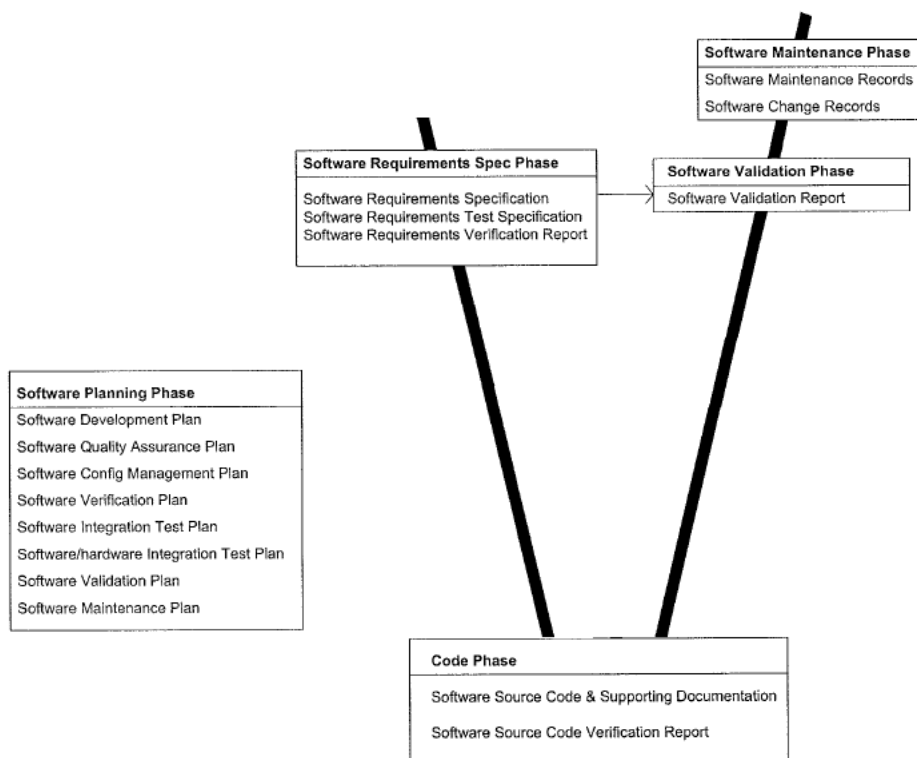
Sekvenssikaavio kuvaa ohjelmistokehityksen elinkaaren, missä vaiheessa ja järjestyksessä dokumentit luodaan sekä mitkä dokumentit luodaan samanaikaisesti (Kuva 13).



**Kuva 14.** EN50128 Ohjelmistokehityksen elinkaari versio 2 [2]

V-mallia (Kuva 14) ja taulukko 2 vertaamalla nähdään, mitkä osat kuvan elinkaaresta ovat halutun SIL tason vaatimia. SIL0 tason ohjelmistokehityksessä pitää standardin mukaan olla dokumentoituna ohjelmiston suunnitteludokumentit (engl. software planning documents), jotka ovat v-mallin suunnitteluvaiheessa (Kuva 14, engl. software planning phase) lueteltuna. Ohjelmiston vaatimus dokumentit v-mallin ohjelmisto vaatimukset vaiheessa (Kuva 14, engl. software requirements spec phase), lähdekoodi & dokumentaation dokumentit esitellään koodi vaiheessa (Kuva 14, engl. code phase), ohjelmiston validointiraportit ohjelmiston validointivaiheessa (Kuva 14, engl. software validation phase), ohjelmiston ylläpitorekisterit ohjelmiston ylläpitovaiheessa (Kuva 14, engl. software maintenance

phase). Kuvassa 15 standardin antamaa v-mallin kuvaa on yksinkertaistettu esittämään vähimmäisvaatimuksia.



*Kuva 15. Ohjelmistokehityksen elinkaari SIL0 dokumentaatio suosituksilla, muokattu lähteestä [2]*

## 2.4 Erot yleisten dokumentaatio sääntöjen ja ketterän ideologian välillä

Perinteinen vesiputousmallin mukainen ohjelmistokehitystyö alkaa ja päättyy dokumentaatioon. Dokumentaation pohjalta aloitetaan ohjelmistonkehitys ja lopputuloksena tuotetaan dokumentaatio siitä mitä saatiin aikaiseksi. Jokainen projektivaihe toteutetaan kokonaisuudessaan loppuun, ennen kuin seuraavaan vaiheeseen edetään eikä aiempiin vaiheisiin voi enää palata. Tämä helpottaa projektin etenemisen seuranta, sekä ohjelman esittelyä asiakkaalle, koska kaikesta on selvä dokumentaatio valmiina jo ennen ohjelmointivaihetta. Huonona puolena sen sijaan on se, että suunnittelussa tapahtuneita vikoja ei pysty korjaamaan eikä kesken projektin voi tehdä muutoksia ohjelmaan. [4, 5, 8]

Ketterässä kehityksessä keskitytään toimintaan. Ideaalitulanteessa kehittäjälle vain sanallisesti kerrotaan, minkälainen ohjelma luodaan. Kun ohjelma on valmis, kehittäjä antaa sen eteenpäin lyhyen suullisen kuvauksen kera. Dokumentaatio on joko lähes olematonta tai pääasiassa koodipohjaista, jota vain ohjelmoijat pystyvät käyttämään. Tämä saattaa olla yritykselle ongelma koska laitevalmistuksen

ja -testauksen työntekijät, organisaation johto ja myyntipuoli eivätkä asiakkaat välttämättä ymmärrä tai pääse käsiksi tähän dokumentaatioon. Hyvinä puolina muutoksia ja korjauksia pystytään tekemään pitkin projektia. Muuttuviin asiakassuhteisiin ja vaatimuksiin voidaan mukautua. Huonona puolena vähäinen dokumentaatio. Koodiin perustuva dokumentaatio ei avaudu muille organisaation jäsenille. Moni laitevalmistaja tekee itse myös ohjelmiston laitteisiinsa, mutta ohjelmointi ja laitevalmistus ovat omat osastonsa. Kattavammasta dokumentaatiosta hyötyvät myös yritysjohto ja myyntipuoli, jotta he tietävät mitä oma yritys tekee ja he pystyvät paremmin keskustelemaan asiakkaiden tai yhteistyökumppanien kanssa yrityksen asioista. [5]

Standardit vaativat enemmän perinteistä dokumentaatiota, kuin dokumentaationa toimivaa koodia. EN 50128 standardin vähimmäisvaatimukset eli SIL0 taso suosittelee viittä perinteistä dokumenttia lähdekoodin lisäksi. Tässä tapauksessa suunnittelu ja vaatimukset käsittävät vain yhden dokumentin ai-  
hetta kohden. [1, 2, 3]

## 2.5 Ketterän kehityksen luomat haasteet dokumentaatiossa

Dokumentoimisen haasteina ketterässä kehityksessä ovat ketterän kehityksen perusluonne, jolle ominainen piirre on suullinen tiedonvälitys sekä ketterän kehityksen monet erilaiset variaatiot [5]. Näiden piirteiden takia ketterälle kehitykselle on vaikeaa luoda tietynlaista kattavaa dokumentointikaavaa, koska dokumentaation vaatimukset vaihtelevat tehtävän työn, yritysrakenteen sekä ketterän kehityksen metodin mukaan.

Haasteita ratkaistaessa pitää keskittyä siihen, mikä on keskeistä sekä dokumentaatiossa että ketterässä kehittämisenä. Dokumentaatiolle keskeistä on se, että projektin etenemisen ja onnistumisen kannalta tärkeät asiat sekä yrityksen tulevaisuuden ja asiakkaan tarpeiden kannalta oleelliset asiat kirjataan ylös. Näitä ovat projektin vaatimukset ja tavoitteet, aikataulutus, suunnittelu, projektidokumentti, sekä asiakkaan vaatimat dokumentit. Ketterälle kehitykselle keskeistä on, että ohjelmistokehittäjät eivät seuraa tarkkaa suunnitelmaa vaan luovat koodia, joka vastaa haluttuja vaatimuksia. Ohjelmistokehitystyö on jaettu iteraatioihin, joissa joko ohjelmaa kehitetään aina hiukan eteenpäin tai luodaan uusia ohjelmistokomponentteja, joista ohjelmakokonaisuus koostuu. Iteraatioiden jälkeen ohjelmisto julkaistaan. Ketterän kehittämisen tyylistä riippuen iteraatioiden ja julkaisujen välillä tehdään esimerkiksi testausta tai esijulkaisu ja päivityksiä. [5]

### 3. YRITYKSEN OHJELMISTODOKUMENTAATION HAASTEET

Yrityksen nykyisen tilanteen selvittämisen pohjana käytetään avainhenkilöiden haastatteluja. Haastattelujen tavoitteena on selvittää nykyiset dokumentaatiotavat, ongelmat, tarpeet, toiveet sekä projektien pullonkaulan.

#### 3.1 Nykyinen käytäntö

Nykytilan selvittämiseksi yrityksen avainhenkilöitä haastateltiin. Haastattelukysymykset sisälsivät kysymyksiä ohjelmistokehityksestä, projektien rakenteesta, yrityksen dokumentaatiokulttuurista ja dokumentaation mahdollisista hyödyistä markkinointiin. Haastateltavia oli seitsemän ja heidän toimenkuvia ovat: Software developer, Project manager, Key accountant manager, Project liaison ja Research and development engineer. Haastattelukysymyksistä tehtiin kirjallinen versio, joka annettiin haastateltavalle haastattelun alussa (Liite A). Keskustelu nauhoitettiin ja vastaukset kirjattiin pääkohdin ylös haastattelulomakkeeseen.

Haastattelukysymykset suunniteltiin siten että haastateltava voi vastata vapaasti oman näkemyksensä mukaan, jotta saadaan todellinen mielipide eikä haastateltavaa ohjata vastaamaan ”oikein”. Vastaukset kysymyksiin eroavat toisistaan hyvin paljon, mutta tuloksia analysoimalla niistä löydettiin yhteneviä näkemyksiä (Taulukot 4-7).

#### 3.2 Haastattelun tuloksia

Merkittävimminä tuloksina haastattelusta saatiin, että yleisellä tasolla ongelmina ovat resurssit ja dokumentaatio. Resurssit ja dokumentaatio nousi neljän kysymyksen vastauksista esille. Esimerkiksi kysymykseen ”Ohjelmisto tuotannon pullonkaula?” seitsemästä haastateltavasta neljä mainitsi resurssien puutteen ja kolme dokumentaation puutteen. Tässä yhteydessä resurssit viittaavat ajallisiin ja työntekijä määrällisiin rajoituksiin. Dokumentaatio eritoten sai monenlaista palautetta: dokumentaatiota ei oikeastaan ole, dokumentaatiolla ei ole yhtä tiettyä paikkaa mihin laittaa / mistä etsiä, dokumentaatioissa ei kerrota miksi on päädytty johonkin ratkaisuun, dokumentaatio ei ole sellaista, josta olisi apua uusille työntekijöille.

Dokumentaation epäselvyyden johdosta ei ollut yllätys, että työntekijöiden mielestä työaika tulisi käyttää enemmän dokumentaatioon. Suurimpana ongelmana on, että asiasta ei ole selkeää yritys politiikkaa.

Haastateltavat edustavat erilaisia toimenkuvia yrityksen sisällä, jonka vuoksi dokumentaation viemä aika työajasta vaihtelee nolasta yli kymmeneen prosenttiin. Valtaosan mielestä dokumentointiin tulisi käyttää nykyistä enemmän aikaa (Taulukko 4, sarake 2). Hyvin moni kokee, että tieto ei pysy

yrityksessä, vaan katoaa sitä mukaan, kun työntekijöitä lähtee talosta. Tiedon hävitessä yrityksestä, vaikeutuu uusien projektien teko. Myyntivaiheessa ajatellaan, että koska myytävää ratkaisua on tehty aikaisemmin, niin tällä kertaa se voidaan tehdä pienemmällä työmäärällä. Projektia tehtäessä törmätään tilanteeseen, että ainut henkilö, joka tietää miten toiminnallisuuksia on tehty, on lähtenyt eikä ole olemassa sopivaa dokumentaatiota kuvaamaan kuinka aikaisemmin on toimittu. Osa työntekijöistä esitti huolensa siitä, että uusilla työntekijöillä ei ole mitään referenssiä minkä pohjalta toimia.

Projektien pullonkaula yleisellä tasolla sai monia erilaisia vastauksia (Taulukko 4, sarake 3). Tässä havaitaan, että toimenkuvasta riippuen erilaiset tilanteet nähdään ongelmakohtina. Projektien sujuvuuden ja mahdollisten säästöjen takia yrityksen kannattaisi sisäisesti tutkia eri projektivaiheiden pullonkaulojen vakavuutta ja mahdollisia ratkaisuja.

**Taulukko 4:** Haastattelun yleiset kysymykset

Dokumentaatio vie työajasta	pitäisikö enemmän työaikaa käyttää dokumentointiin?	Projektien yleinen pullonkaula, joka syö aikaa?	Mikä on olennaista käsitellä tässä työssä?
0%: (1)	Kyllä: 66,6% (4)	ongelmien selvitys: (2)	dokumentti miksi ratkaistun näin, vaikka ei olisi virallinen dokumentti
1-5%: (2)	Ei osaa sanoa: 0% (0)	Specsin luonti: (1)	Vaatimukset (asiakas ja sisäiset), Toiminnallisuudet.
6-10%: (2)	Ei: 33,3% (2)	Projektin aloitus/kick off: (2)	Projektin prosessit, mihin softa laitetaan / mitä sillä tehdään kun se on tehty
yli 10% (1)		Resurssipula: (1)	
eivät osaa sanoa: (1)			

Kaksi seitsemästä haastateltavasta uskoo, että asiakkaiden mielestä dokumentaatio on hyvää ja osan mukaan se riippuu asiakkaasta. Osa asiakkaista on hyvin vaativia dokumentaation suhteen. Haastattelussa nousi esille myös, että dokumentaatiota tehdään vain asiakasta varten.

Asiakkaat asettavat dokumentaatiolle pääasiassa aikataulullisia vaatimuksia.

Lähes kaikkien mukaan asiakasvaatimukset täytetään, mutta haastattelussa nousee esille, että tekniset vaatimukset täytetään aina, mutta aikataulullisia ei (Taulukko 5, sarake 3).

**Taulukko 5: Haastattelun myynnin kysymykset**

Mitä mieltä asiakkaat ovat mitronin dokumentaatiosta?	Minkälaisia vaatimuksia asiakkaat asettavat dokumentaatiolle?	Päästäänkö näihin tavoitteisiin?	Vaikuttaako Mitronin dokumentaatio asiakaskuntaan?	Minkä tason dokumentaatio on myyjille hyödyllistä?	Voiko dokumentaatio vaikuttaa markkinointiin?
Riippuu asiakkaasta (2)	Aikataulu (4)	Vaatimukset täytetään (4)	Kyllä (3)	Platform (2)	Kyllä (6)
Hyvää (2)	Tyyppi (4)	Myöhästyy (2)	Dokumentaation puute voi vaikuttaa negatiivisesti (2)	Tekniset kuvaukset (4)	Platform dokkarit helpottaa (2)
Vain asiakas dokumentaatiota (2)			Yksi kilpailukyvn mittareista (1)	Myyjät myy mitä tulee mieleen, koska eivät tiedä mitä meillä on (1)	Tekninen näkökulma ei välttämättä aukee kaikille (1)

Ohjelmistotuotannon pullonkauloissa kaikki haastateltavat luettelivat useamman kuin vain yhden pullonkaulan (Taulukko 6, sarake 1). Suurimpana pullonkaulana koettiin resurssien puute eritoten lyhyet aikataulut ja työvoimarajoitteet (4/7). Työvoimaraajoitteisiin vaikuttaa oleellisesti se, että osaaminen on vain tietyillä henkilöillä, jotka ovat sidottuna jo muihin projekteihin (3/7). Dokumentaation puute koettiin myös pullonkaulana (3/7). Dokumentaation puute liittyy resurssipulaan siten, että kun tietotaito kertyy vain tietyille henkilöille, eikä muilla ole pääsyä näihin tietoihin, ei kyseessä olevalta osaamisalueelta synny uusia osaajia. Tästä johtuen ongelma periytyy projektilta toiselle ja työntekijät erikoistuvat tekemään vain niitä tehtäviä, joita muut eivät osaa.

Ongelmallisimpana ohjelmistotuotannossa pidettiin vaatimustenhallintaa (Taulukko 6, sarake 2). Ohjelmistotuotannon vaatimuksissa olevia epäselvyyksiä ja väärinymmärryksiä pidettiin yleisenä ja työllistävimpänä ongelmana (4/7). Työtunteja menee hukkaan, kun vaatimuksia ei heti aluksi ole tehty tarpeeksi selkeästi vaan on jäänyt tulkinnan varaa, tai asioita on suoranaisesti ymmärretty väärin. Haastatteluissa ilmeni, että on ollut tapauksia, joissa on luultu mitä asiakas haluaa, mutta luulo ei ole vastannut todellisuutta. Dokumentaatio koettiin myös ongelmallisena (3/7). Haastatteluissa nousi esille, että yrityksellä ei ole varsinaista linjausta dokumentoinnin suhteen. Tästä johtuen tärkeitä asioita jää dokumentoimatta, dokumenttien sijainti ei ole tiedossa eikä oikein edes tiedetä mitä olisi hyvä dokumentoida ja miten. Dokumentaatio pirstaloituu eri paikkoihin ja dokumenteista ei tiedä ovatko ne ajan tasalla vai eivät. Resurssit, etenkin työvoiman suhteen koettiin myös ongelmana (1/7). Tiettyjen henkilöiden osaamista tarvitaan useassa projektissa ja projekteilla itsellään ei aina ole selvästi määriteltyä työryhmää vaan työt rönsyilevät.

Vanhaan tai muiden tekemään koodiin suhtaudutaan hyvin, jos dokumentaatio on kohdillaan (Taulukko 6, sarake 3). Jos dokumentaatio on huonoa ja tekijä on lähtenyt talosta ei muutoksia vanhaan koodiin haluta tehdä (2/7). ”Tehdään alusta” -menteliteetti kohdistuu pääasiassa pieniin ohjelmistoihin, mutta sitä kohdistuu myös koodiin, josta ei ole dokumentaatiota ja tekijä on lähtenyt firmasta (3/7).

Apua koodin ymmärtämiseen haetaan pääasiassa koodissa olevasta kommentoinnista (4/7) (Taulukko 6, sarake 4). Haastattelun yhteydessä selvisi, että suuri ongelma koodissa olevassa kommentoinnissa on se, että niistä ei selviä miksi näin on tehty. Monesti koodin kommentoinnissa käsitellään mitä koodi

tekee, mutta pätevä ohjelmistokehittäjä osaa tulkita koodin toiminnan itse koodista, mutta jos ei ymmärrä miksi näin on tehty ja menee muuttamaan koodia, voi rikkoa loput koodista. Vain yksi haastateltavista kertoi pyrkivänsä dokumentoimaan wikisivuille tekemiään koodeja/ohjelmistoja. Apua haetaan myös työkavereilta kysymällä (2/7).

Ohjelmistotuotannossa projektin dokumentointiin liittyvät vastaukset heijastavat täysin edellisen kohdan vastauksia (Taulukko 6, sarake 5). Suurin osa dokumentoi koodiin (4/7), vain yksi tunnusti dokumentoivansa wikiin ja kaksi kyselyyn vastanneista osallistuu virallisten dokumenttien luontiin. Tässä tapauksessa virallinen dokumentointi tarkoittaa vain asiakasdokumentaatiota.

Haastatteluun vastanneiden mukaan tilanteita, joissa ei tiedä miten jokin toiminnallisuus on koodattu eikä löydä mitään ohjeita, sattuu osalle usein (2/7) ja osalle silloin tällöin (2/7).

**Taulukko 6:** Haastattelun ohjelmistotuotanto kysymykset

Ohjelmisto tuotannon pullonkaula?	Ongelmallisinta ohjelmistotuotannossa?	Vanhaan/muiden koodiin suhtautuminen	Mistä haet apua koodin ymmärtämiseen?	Miten dokumentoit projektia tehdessä?	Törmäätkö tilanteeseen, ettet tiedä miten jokin toiminnallisuus on koodattu eikä löydä mistään ohjeita?
Resurssien puute (4)	Vaatimustenhallinta (4)	Talostalähteneiden koodiin nihkeästi (2)	Wikistä (1)	Koodiin (4)	Usein (2)
Dokumentaation puute (3)	Dokumentaatio (3)	Jos hyvä dokumentaatio etenkin rajapinnoista niin hyvin suhtaudutaan (1)	Kommentoitu koodiin (4)	Wikiin (1)	Sillon tällöin (2)
Osaaminen vain tietyillä henkilöillä: (3)	Resurssit (1)	Tehdään alusta (3)	Kysytään kaverilta (2)	Virallisia dokumentteja (2)	En osaa sanoa (2)

Haastateltavien mukaan projektit suunnitellaan asiakasvaatimusten ja asiakkaan aikataulun mukaan. Projektille luodaan virstanpylväät (engl. milestone / gatet), jotka pitää toteuttaa projektin edetessä.

Tilanteita, joissa ihmetellään, ”kuinka tähän tilanteeseen päädyttiin” on vastanneiden mukaan monesti (4/7), mutta projektista riippuen (3/7) (Taulukko 7, sarake 2). Vastanneet eivät kuitenkaan osanneet sanoa onko jäänyt mieleen, että tässä kohtaa menttiin viimeksikin väärille urille, mutta useamman henkilön mukaan nämä tilanteet johtuvat resursseista.

Lähes kaikkien vastanneiden mielestä jäljitettävyyys on tärkeää (6/7) (Taulukko 7, sarake 3). Asiakkaat vaativat tietyn tason jäljitettävyyttä, mutta jäljitettävyyys koettiin myös tärkeäksi, jotta tietoa voidaan hyödyntää tulevissa projekteissa.

Tärkeiksi tilanteiksi/kohdiksi jotka pitäisi dokumentoida nousivat esille vaatimukset (2/7), muutokset (3/7) ja testaus (2/7) (Taulukko 7, sarake 4).

Haastateltavien mukaan tärkein dokumentaatio tyyppi on, mitä tehtiin ja mihin päädyttiin sekä muutokset (3/7) (Taulukko 7, sarake 5)

**Taulukko 7: Haastattelun projektikysymykset**

Kuinka projektit suunnitellaan?	Onko tilanteita joissa ihmetellään, kuinka tähän tilanteeseen päädyttiin?	Onko projektien jäljitettävyyttä tärkeää?	tilanteita/kohtia joissa pitää dokumentoida?	Minkälainen dokumentaatio on tärkeintä?	Muuta lisättävää?
Asiakas vaatimukset (6)	Monesti (4)	On (6)	Vaatimukset (2)	mitä tehtiin, mihin päädyttiin (3)	softa toiminnallisuudet pitäisi dokumentoida paremmin aikaa tehdä dokumentaatiota, aikataulut liian tiukkaa
Asiakkaan aikataulu (6)	Joissakin projekteissa (3)	Asiakas vaatii (1) Hyödyttää seuraavia projekteja (3)	Muutokset (3)	Muutokset (1)	Eri henkilöillä niin eri tarpeet vs esim softa dokumentaatio
Milestonet / Gatet (2)	Johtuu resursseista (3)		Testaus (2)		

### 3.3 Ongelmat

Haastattelun tuloksista huomataan, että dokumentaatiota tehdään pääasiassa asiakasta, eikä yritystä itseään varten. Tällöin pyritään vain täyttämään vaatimukset ja dokumentaatioissa keskitytään saamaan tekstiä vaadittuihin kohtiin.

Ohjelmoijat dokumentoivat koodiin tai wikiin omasta vapaasta tahdosta. Ketterän kehitysideoideologian mukaan iteraatio-kokouksia pidetään, mutta niistä ei jää tulevia projekteja varten mitään ylös.

Dokumentaation puutteen ja henkilöstön vaihtuvuuden takia aiemmin tehdyt saman tyyppiset projektit eivät auta tulevia projekteja millään tavalla. Tieto/taito katoaa työntekijöiden lähtiessä. Uudet työntekijät eivät löydä apua tai mallia, miten asioita on tehty tai kuinka niitä tässä yrityksessä tulisi tehdä.

Tieto ja dokumentaatio hajautuu projektikohtaisesti eri henkilöille ja eri paikkoihin. Dokumentaation osalta ei ole mitään keskitettyä paikkaa mihin tallentaa ja mistä etsiä tietoa. Tiedon hakuun hukkaantuu paljon aikaa ja vaikka tietoa löytyisi niin ei tiedä onko vielä lisätietoa saatavilla jossain muualla. Tästä syntyy vaikea tilanne päättää, milloin jatkaa tiedon etsimistä ja milloin todeta, että kaikki saatavilla oleva tieto on jo käytössä.

### 3.4 Dokumentaation tarpeet

Haastatteluiden perusteella yrityksen dokumentaatio perustuu lähinnä siihen mitä asiakas vaatii dokumentoitavaksi, mutta tämä ei monelta osin hyödytä yritystä, koska asiakkaan vaatima dokumentaatio keskittyy asiakkaan tarpeisiin projektilta ja asiakkaan vaatimusten täyttämiseen. Yrityskehityksen kannalta olisi tärkeämpää dokumentoida miten ja miksi eri projekteissa ja projektien tilanteissa toimittiin, jotta tulevissa projekteissa voidaan hyödyntää aiemmin tehtyä ja uudet työntekijät saavat referenssejä siihen, miten asioita on aiemmin tehty. Tekniikan alan yrityksissä on yleinen ongelma, jossa myyjät myyvät jotakin mitä he luulevat yrityksellä olevan. Se johtaa tilanteeseen, jossa jo myytyjä ominaisuuksia pitää luoda uutena, vaikka tietotaitoa ei olisi. Tuotekehityksen pitäisi olla yrityksen oma selvä visio, miten parannetaan ja minne yritys on menossa, eikä tilausuhan alla tehtyjä uusia tuotteita.

Dokumentaation toimivuuteen vaikuttaa keskeisesti myös sijainti ja ajantasaisuus. Sijainti tulisi olla keskitetty ja looginen, jotta tiedon etsintä pystytään rajoittamaan tietylle alueelle. Yrityksen wiki-sivut tai erikoistunut dokumentaatio-ohjelmisto kuten esimerkiksi Polarion<sup>1</sup> tai Jira<sup>2</sup> ovat hyviä paikkoja keskittää dokumentaatio.

Esimerkki dokumentaation käytöstä wikissä:

- Wikiin luodaan projektille sivu, jossa kerrotaan yleisesti kenelle tehdään (asiakas), mitä on sovittu tehtäväksi ja milloin pitäisi olla valmis
- Tehdään wiki-sivulle linkki sivuun, jossa on projektisuunnittelu
- Tehdään wiki-sivulle linkki sivuun, johon luodaan iteraatiot
- Iteraatio-sivuille kirjataan mitä kussakin iteraatiokierrossa pitäisi tehdä ja mitä lopulta tehtiin. Jokainen ohjelmoija kirjoittaa sivulle suurpiirteisesti mitä toiminnallisuuksia teki työstettäväänsä koodiin ja lataa sivulle ko. koodin.
  - o Tällöin sivu kertoo mitä tehtiin ja miksi
  - o Koodista ja koodin dokumentaatiosta voi katsoa tarkasti miten

Jotta dokumentaatio pysyisi ajankohtaisena ja tieto tulisi keskitettyä tiettyyn paikkaan olisi ketterän kehityksen kannalta parasta luoda dokumentaatiota jokaisen iteraation (engl. sprint) jälkeen. Kun ensimmäinen iteraatio päättyy, pidetään kokous, jossa käydään läpi mitä piti saada aikaiseksi, mitä saatiin aikaiseksi ja kaikki kirjataan ylös.

---

<sup>1</sup> Polarion on projektien ohjaus ja hallinta ohjelmisto <https://polarion.plm.automation.siemens.com/products/polarion-alm>

<sup>2</sup> Jira on projektien ohjaus ja hallinta ohjelmisto <https://www.atlassian.com/software/jira>

## 4. TEORIAN SOVELTAMINEN KÄYTÄNTÖÖN

Tässä kappaleessa käsitellään diplomityössä tehtyjä IEEE 1558 Standardin mukaisesti luotuja ohjelmiston vaatimus spesifikaatio (engl. Software Requirements Specification, SRS), ohjelmiston rakenteen kuvaus (engl. Software Design Description, SDD) ja ohjelmiston testaustapa (engl. Software Test Procedure, STPr) dokumentteja. Dokumenttien luomisessa hyödynnettiin Polarion dokumentaatio-ohjelmaa, johon Taipuva Consulting Oy tarjosi tutkimuslisenssin. Olennaisena osana tutkitaan, miten Polarion:ia voi hyödyntää dokumentaation luomisessa, jakamisessa sekä säilyttämisessä.

### 4.1 Toteutus ja tulokset

Yhtenä toimivan dokumentaation edellytyksenä on keskitetty sijainti. Dokumentaation sijainti pysyy yhtenäisenä, jos dokumentaatio tehdään ja säilytetään siihen kuuluvassa järjestelmässä. Hyviä esimerkkejä tämän tyyppisistä järjestelmistä ovat Jira ja Polarion.

### 4.2 Polarion

Polarion on hyvä työväline dokumentaation säilöntään ja luomiseen. Polarion:iin voidaan luoda omat projektit. Projekteille on monia erilaisia pohjia valmiina mm. ohjelmistokehitys, järjestelmäkehitys jne. Parhaita puolia Polarion:ssa on se, että dokumentteja voidaan tuoda (engl. import) järjestelmään ja tuontivaiheessa voidaan dokumentista kerätä niin sanottuja työkappaleita (engl. work item). Työkappaleiden keräystä varten asetetaan järjestelmälle ehdot, missä kappaleessa ja minkälaisia sanoja työkappaleissa käytetään ja Polarion hakee sinulle dokumentista valmiiksi kaikki työkappaleet omiksi erillisiksi työkappalepalikoiksi, joita voit hyödyntää tai suoraan viitata muihin projektidokumentteihin. Tämän toiminnon avulla voidaan automaattisesti esitäyttää tulevia dokumentteja, joissa viitataan muiden dokumenttien osiin.

#### 4.2.1 Projektit

Polarion:issa luodaan projekti, johon dokumentaatio tuotetaan. Projektipohjia (engl. project template) on yhdenkäsän erilaista, joista neljä on ohjelmointi-, kolme järjestelmä-, yksi spesifikaatio- ja yksi sääasemapohja. Pohjaa ei ole pakko valita, mutta pohjat sisältävät paljon hyödyllisiä dokumentaatiokenteita. Esimerkiksi system engineering project -pohjassa on valmiina omat kansiot Concept, Requirements, Design, Risks, Development ja Maintenance. Kansioihin voidaan luoda tai tuoda (engl. import) dokumentaatiota.

#### 4.2.2 Tiedostojen tuonti ja vienti

Tiedoston tuomiseksi (engl. import) valitaan dokumentti, jonka halutaan tuoda Polarion:iin. Tuominen tapahtuu projektin kansion etusivulta, jolloin tuotava dokumentti tulee siihen kansioon, josta tuominen

on valittu. Tiedostoa tuotaessa voidaan automaattisesti luoda työkappaleita rajaamalla tuonti toiminnolle, minkälaisista dokumentin osista halutaan työkappaleita. Valintaa voidaan rajata otsikon avulla sekä tekstisisällön avulla. Esimerkiksi valitaan 3.2 Features. Asetetaan ehdoiksi "is heading" ja "contains words (a=A): Functional requirement" sekä kohtaan sisällytetään seuraava kappale työkappalekuvaukseksi "isn't heading". Tällöin tuonti -toiminto etsii Features otsikon alta Functional requirement otsikot, joista tehdään automaattisesti työkappaleita ja näiden alla oleva kappale asetetaan työkappaleen kuvaukseksi. Jos dokumentin ID:n alta tyyppi -kohdasta valitaan Requirements specification ja "Mark paragraphs" kohdan alta valitaan System Requirement, saadaan Functional requirement:it automaattisesti määritettyä järjestelmävaatimuksiksi (Kuva 16, 17).

**Import Document 1487 - AT Citadis Spirit IPEP SRS (Review\_1).docx**

Title: 1487 - AT Citadis Spirit IPEP SRS  
 Name (ID): 1487 - AT Citadis Spirit IPEP SRS  
 Type: Requirements Specifiicator  
 Space: Requirements

+ Add Work Item Rule

**Mark paragraphs from** 3.2 Features

as System Requirer if all of the conditions are met:

is heading  
 contains words (a=A) Functional requirement

Show advanced options

**Include the next paragraph in the Work Item description**

if any of the conditions are met:

isn't heading

**Perform following actions**

Set field -- not selected --

+ Add

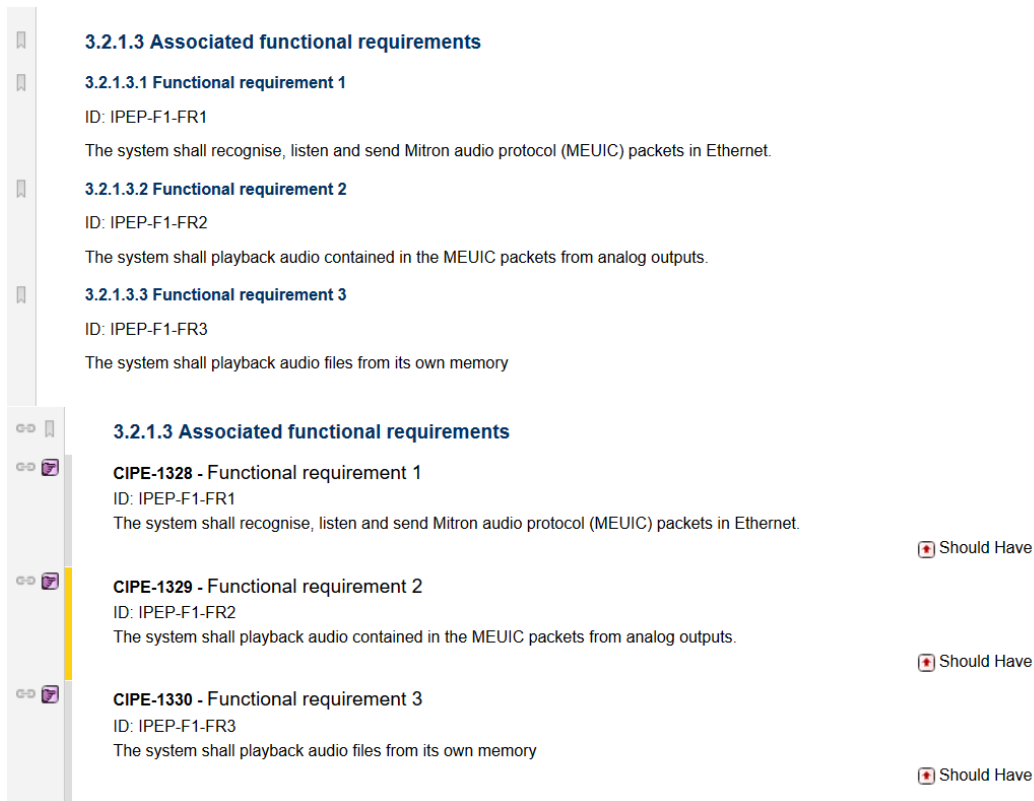
**Additional options**

Import first paragraph as title of the Work Item

Fields at start id  
 Fields at end severity

<< PREVIEW Cancel import IMPORT >>

**Kuva 16.** SRS dokumentin tuominen (engl. import) ja järjestelmä vaatimusten (engl. system requirement) automaattinen luominen



*Kuva 17. Polarion:iin tuotu dokumentti otsikko tunnisteella (yläpuolella) ja automaattisella järjestelmä vaatimusten luomisella (alapuolella)*

Polarion vie (engl. export) tiedostot PDF-muodossa. PDF-muotoisten tiedostojen tekstisisältöä ei voi muokata. Tästä johtuen yritysdokumentaatiota voi muokata vain henkilö, jolla on käyttäjätunnus sekä oikeudet käyttää yrityksen Polarion:ia.

## 4.2.3 Työkappaleet

Työkappaleet ovat dokumentaation osia, joita voi kopioida, viitata tai muuten käsitellä muissakin dokumenteissa tai työnjaossa. Erilaisia työkappaleluokkia on 14. Jos työkappaletta luodessa ei määrittele luokkaa niin työkappaleesta tulee tehtävä (engl. task). Muita luokkia ovat System Requirements, System Test Case, Software Requirement, Software Test Case, Mechanical Requirement, Mechanical Test Case, Electrical Requirement, Electrical Test Case, Risk, Change Request, Work Package, Task, Issue, Unit Test Case.

## 4.2.4 Uudelleenkäytä ja haarauta

Dokumentteja voi uudelleenkäyttää kahdella eri tavalla. Uudelleenkäytä (engl. reuse) luo erillisen kopion, jossa alkuperäisen dokumentin lisäksi sen työkappaleista tehdään duplikaattiversiot kopioon.

Haarautus (engl. branch) kopiesa työkappaleet ovat referenssejä alkuperäisen dokumentin työkappaleisiin. Työkappaleita ei voi muuttaa kopiesa, mutta jos alkuperäisessä dokumentissa työkappaleita muuttaa, tulee muutos haarakopion työkappaleisiin. Haarakopion työkappaleet voi asettaa jäädytys (engl. freeze) -tilaan, jolloin alkuperäisiin työkappaleisiin tehdyt muutokset eivät päivity haaran työkappaleisiin.

### **4.3 Case Study IPEP**

Asiakas toivoi lisädokumentaatiota uusimpaan projektiin. Uudet dokumentit ovat SDD, SRS, STPr dokumentit, jotka luotiin diplomityössä. Dokumenttien tulee olla IEEE 1558 standardin mukaisia. Näiden dokumenttien lisäksi tutkitaan kuinka dokumentaatiota voi tehdä helpommin ja tehokkaammin avustavien ohjelmien, tässä tapauksessa Polarion:in avustuksella. Luontiprosessissa ilmeni myös, missä vaiheessa projektia kukin dokumenteista tulisi luoda, jotta niiden luominen olisi projektin kannalta luontevaa, tehokasta ja projektia tukevaa.

#### **4.3.1 SDD**

Software Design Description on IEEE 1558 standardin vaatima dokumentti, jossa määritellään ohjelmistosuunnittelu päätökset. Tämän tyyppisiä päätöksiä ovat esimerkiksi ohjelmiston käyttäytymisen suunnittelu sekä ohjelmistokomponenttien valinta ja rakenne. SDD dokumentti sisältää myös kuvauksen ohjelmistoarkkitehtuurin suunnittelusta. Dokumentin tärkeä tehtävä on antaa asiakkaalle näkyvyyttä suunnittelusta ja tarjota ohjelmiston ylläpitoon tarvittavaa tietoa. [3]

#### **4.3.2 SRS**

Software Requirements Specification on IEEE 1558 standardin vaatima dokumentti, jossa ohjelmistovaatimukset käsitellään siten että, jokaisen vaatimuksen voi objektiivisesti verifioida ja validoida dokumentissa esitetyin keinoin. SRS dokumentissa pitää myös esittää ohjelmiston implementointiin liittyvät oletukset ja suunnittelun luomat rajoitukset. [3]

#### **4.3.3 STPr**

Software Test Protocol dokumentti tarjoaa SCI hyväksyttämisen testauksessa käytettävät erityisproseduurit askeleittain sekä testaukseen liittyvät erityisvaatimukset. [3]

#### **4.3.4 Dokumentaatio**

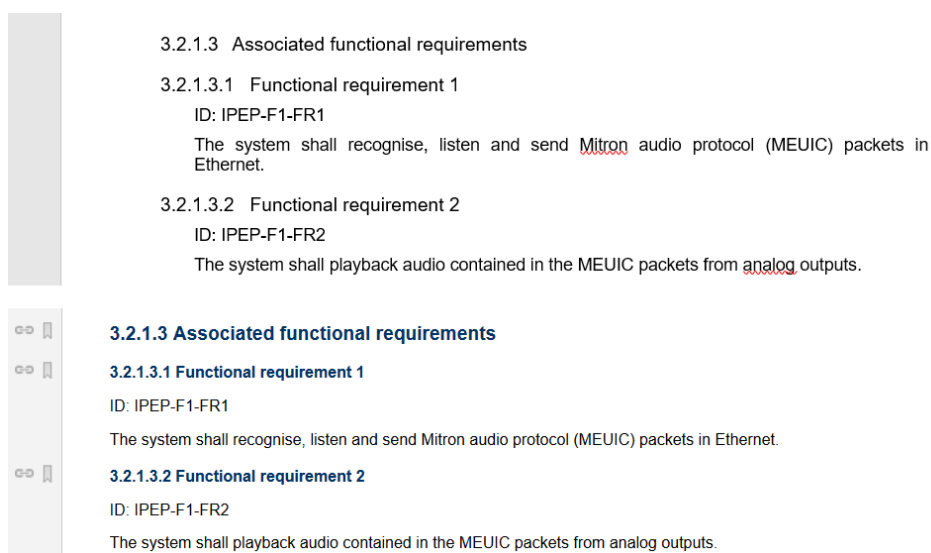
Vaaditut dokumentit luotiin ensin perinteisinä word-tiedostoina. Dokumentit luotiin samalla pohjalla, kuin aikaisemmatkin projektit. Word dokumenttien pohjalta aloitettiin Polarion:in testaaminen dokumentaatio käyttöön.

### 4.3.5 Polarion dokumenttien luomisessa

Ohjelma tarjoaa omat projektipohjat, joissa on valmiina kansiot erilaisiin dokumentaatiotarpeisiin. Tässä tapauksessa käytettiin System Engineering Project, koska se sisältää kansiot Requirements ja Design. Kun dokumentaatio tehdään vähintään Polarion:iin, niin dokumentaatiolla on yksi keskitetty paikka, josta kaikki löytyy. Jos dokumentaatio tehdään Polarion:issa, niin sitä ei myöskään voi muuttaa kuin Polarion:issa, koska dokumentit voi viedä ulos (engl. export) vain PDF-tiedostona.

### 4.3.6 Polarion dokumenttien tuonnissa

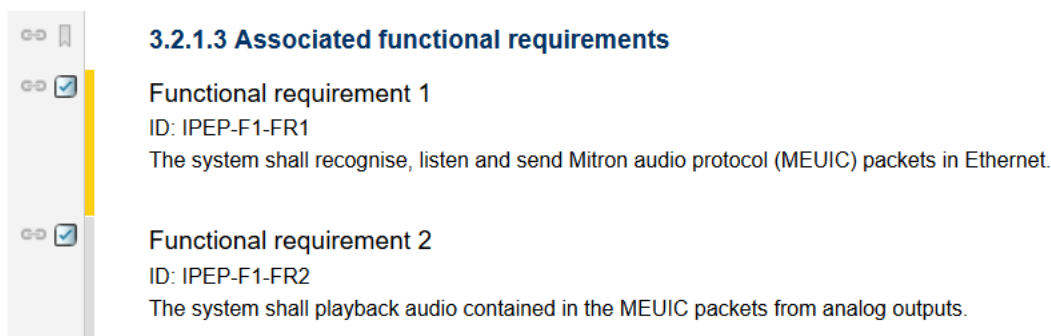
Polarion:iin tuoduissa word-dokumenteissa ilmenee rakenteellisia virheitä. LIST OF FIGURES ja LIST OF TABLES listoista näkyy vain otsikko, mutta ei sisältöä. Sisällysluettelon sivunumerot eivät näy, otsikoiden koot ja fontin väri muuttuvat (Kuva 18).



**Kuva 18.** SRS dokumentti Word:ssä ja Polarion:ssa

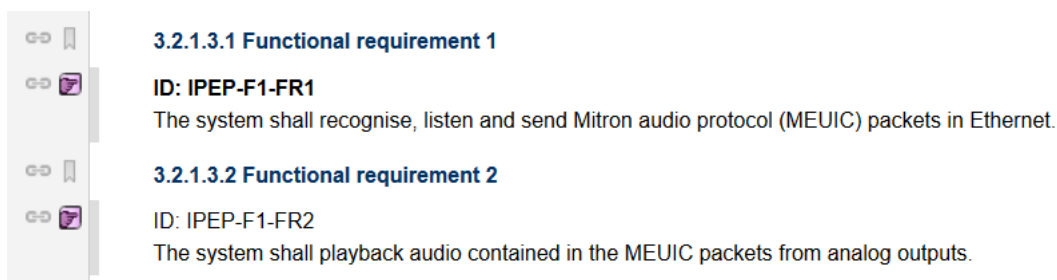
Tiedostoa tuodessa voi antaa rajaukset minkä mukaan rajaus tekee automaattisesti dokumentin osista työkappaleita. Ohjelmistovaatimus spesifikaatiosta (SRS) saatiin automaattisesti tehtyä toiminnallisista vaatimuksista työkappaleita. Näitä työkappaleita hyödynnettiin toiminnallisen määrittelyn luomisessa.

Dokumenttiosien muuttaminen työkappaleiksi hävittää tekstiasetukset. Esimerkiksi alaotsikosta Functional requirement 1 ja sen kappaleesta tehty työkappale Functional requirement 1, menettää otsikkoasettelun sekä kappaleen sisennyksen (Kuva 18,19).



*Kuva 19. Tuonti ja automaattinen työkappaleiden luonti*

Ongelman voi ohittaa tekemällä työkappaleen vain alla olevasta kappaleesta. Tällöin työkappaleen tunnistamiseksi valitaan työkappaleen nimeksi toiminta vaatimuksen (engl. functional requirement) ID ja loput kappaleesta asetetaan työkappaleen kommentiksi (kuva 19).

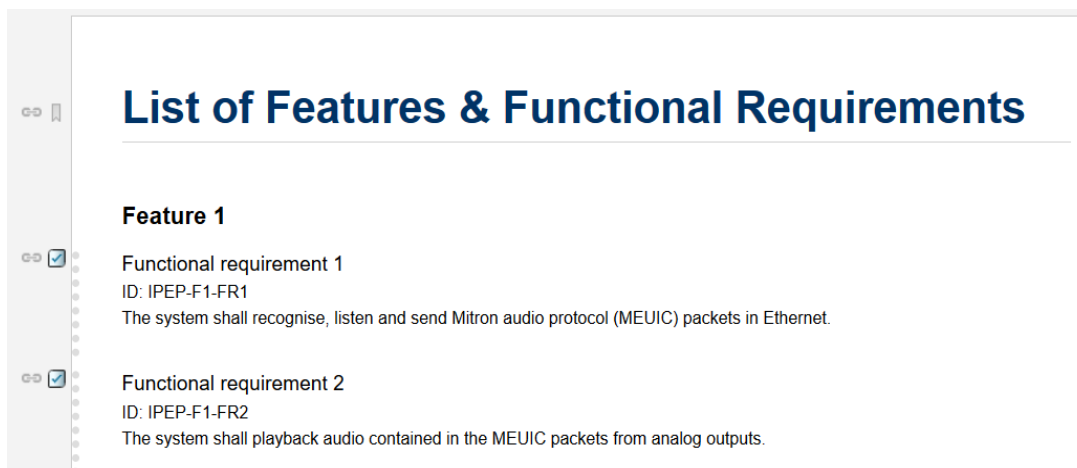


*Kuva 20. SRS dokumentista tehty system requirement, otsikolla ja kommenttiosilla sekä vain kommenttiosilla*

Kun tiedosto tuodaan Polarion:iin, voidaan kappaleista tehdä järjestelmävaatimukset (engl. system requirement). Tällöin dokumenttiin ilmestyy järjestelmävaatimuskoodi. Se rikkoo dokumentin rakennetta ja asettelua, mutta osoittaa helposti, kuinka järjestelmästä löytää kyseisen järjestelmävaatimuksen (Kuva 20).

#### 4.3.7 Dokumenttien luominen työkappaleista

SRS dokumentin toiminnallisista vaatimuksista (engl. functional requirement) tehtyjä järjestelmävaatimuksia (engl. system requirement) käytettiin List of feature and functional requirements -dokumentin luomiseen. Järjestelmävaatimukset voidaan hakea joko työkappaleiden listasta, tai ID:n avulla (Kuva 21).



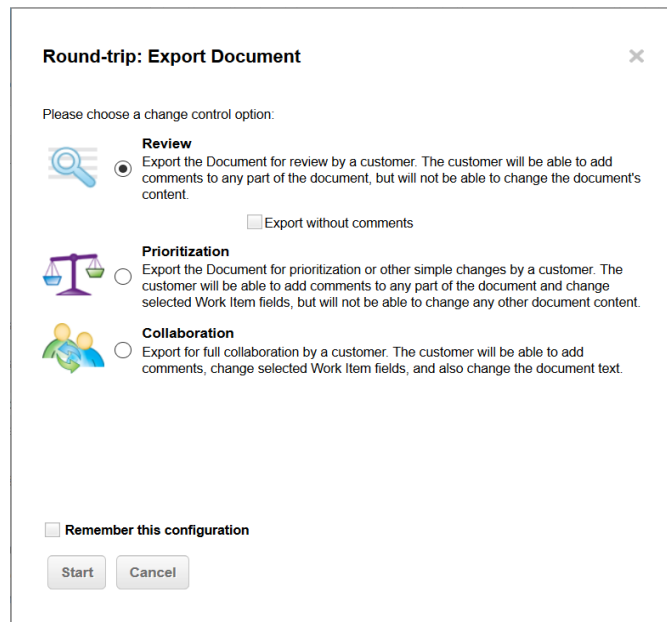
*Kuva 21. Työkappaleista luotu dokumentti*

### 4.3.8 Vienti

Polarion:in oma dokumenttinäkymä ei näytä sivujakoja, minkä vuoksi kunnollisen rakenteen omaavan dokumentin vienti (engl. export) vaatii useita yritys- ja erehdys-kertoja. Vientiformaatteja on kolme erilaista: pdf, Word ja ReqIF.

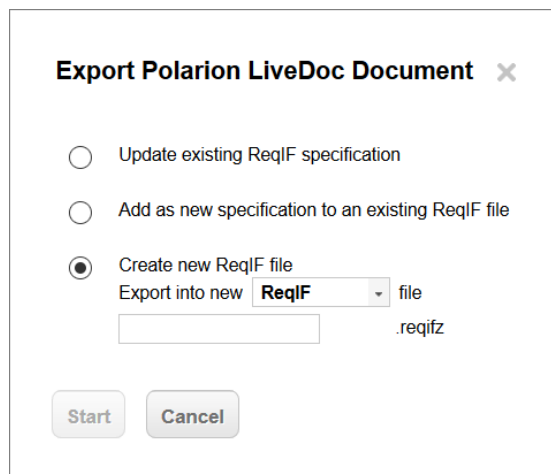
Pdf-muotoon vienti on helppoa, koska tietomuoto on yksinkertainen eikä pdf-tiedostoa voi muuttaa. Ainoat valintavaihtoehdot ovat paperin koko, vaaka vai pystysuuntaan, sivun leveyteen sovittaminen, kirjanmerkkien generointi sekä ylä- ja alatunnisteen sisällyttäminen. Oletusasetukset ovat tavallisen dokumentin viemiseen asetettu, joten normaalisti voi vain painaa Export.

Word-muodossa saa kolme vaihtoehtoa: Review, Prioritization ja Collaboration (Kuva 22). Review-muodossa dokumenttiin ei voi lisätä muuta kuin kommentteja, jos kommenttien lisääminen on sallittu. Prioritization-muodossa dokumentin työkappaleissa muuttamiseen sallittuja osia, mutta työkappaleen sisältöä ei voi muuttaa. Tässä muodossa on myös mahdollista lisätä kommentteja. Collaboration-muodossa pystyy lisäämään kommentteja, muuttamaan työkappaleiden sallittuja osia ja muuttamaan dokumentin tekstiä.



**Kuva 22.** Word dokumentin Export toiminto valintoineen.

ReqIF eli Requirements Interchange Format on XML tiedostomuoto, jota käytetään vaatimusten käsittelyyn. Sillä viedään (engl. export) vain otsikko ja vaatimukset. Luodun XML-tiedoston sisältöä voidaan päivittää ReqIF export toiminnon update existing ReqIF specification (Kuva 23).



**Kuva 23.** ReqIF Export valinnat

## 4.4 Dokumenttipohjat

Tulevia projekteja varten tehtiin standardin mukaiset dokumenttipohjat. Uusissa projekteissa tiedot voi suoraan täyttää pohjassa olevien otsikoiden alle. Dokumenttipohjille tehtiin oma projekti-pohja, jotta kaikille dokumenttipohjille on yksi keskitetty paikka, josta niitä voi hakea. Pohjat voi kopioida projektista projektiin uudelleen-käytä (engl. reuse) -toiminnon avulla (Kuva 24). Uudelleenkäyttö -toiminnossa voit määrittää dokumentille nimen, projektin johon kopioidaan ja mihin projektikansioon kopio laitetaan.

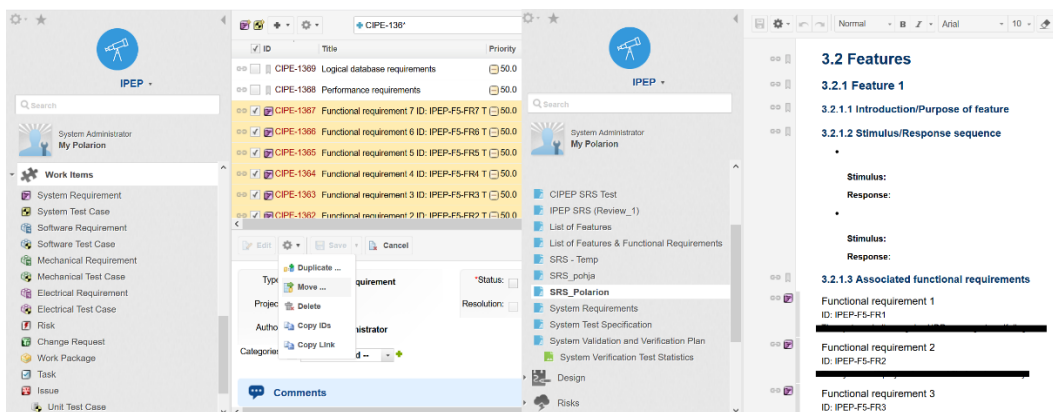
Dokumenttipohjien ansiosta uusissa projekteissa muutamien dokumenttien osia ei tarvitse uudestaan kirjoittaa. Aikaisemmassa kappaleessa käsiteltiin dokumenttien luomista työkappaleista.

*Kuva 24. Reuse toiminto*

## 4.5 Yritysdokumentaation päivitys standardin mukaiseksi

Polarion ei tarjoa toimintoa, jolla generoidaan vanhasta dokumentaatiosta uutta. Yritysdokumentaation muutto rautatiestandardin mukaiseksi pitää tehdä lähes käsityönä. Muutosta varten luotiin IEEE 1558 mukaiset pohjat SRS, SDD, STRM ja STPR dokumenteille (Liite B, C, D, E). EN 50128 ei tarkalleen määritä vaadittujen dokumenttien rakennetta tai sisältöä, minkä takia sitä ei tässä huomioitu.

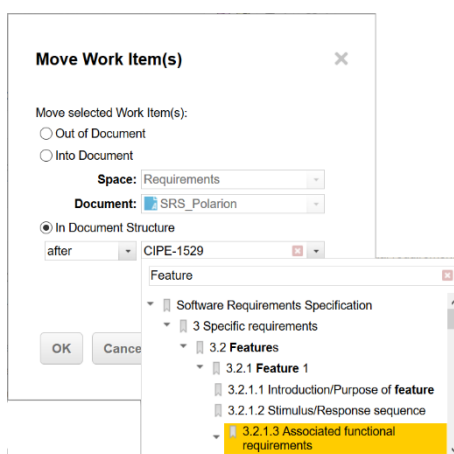
SRS\_pohja:sta tehtiin haarakopio, joka nimettiin SRS\_Polarion:iksi esimerkkiä varten. Yritysdokumentaatio SRS\_Mitron ladataan Polarion:iin siten, että pohjassa käsiteltävät työkappaleet kerätään dokumentista. Haarakopion otsikoiden sisällöt haetaan järjestelmään ladatuista työkappaleista. Kappaleet haetaan työkappale valikosta kappale koodin avulla (Kuva 25). Esimerkiksi vaatimuksia haakiessa käytettiin CIPE-136\* haku koodia, joka tuo esiin kaikki CIPE-1360 ja CIPE-1369 välillä olevat toiminta vaatimukset (engl. Functional requirement).



*Kuva 25. Työkappale valikko ja dokumenttiin tuodut vaatimukset*

Toimintavaatimus työkappaleet pitää ensin siirtää siirto komennolla (engl. move) SRS\_Polarion:iin (Kuva 25). Dokumenttiin siirrossa siirto komennon ikkunasta valitaan dokumenttiin asetus (engl. Into Document), tila kohdasta (engl. Space) valitaan missä SRS\_Polarion dokumentti sijaitsee ja dokumentti (engl. Document) kohdasta valitaan työstettävä dokumentti SRS\_Polarion (Kuva 26). Edellä mainitut toimenpiteet siirtävät työkappaleet SRS\_Polarion dokumentin loppuun. Samat työkappaleet valitaan uudelleen ja siirto komentoa käytetään uudelleen. Tällä kertaa valitaan dokumentin rakenteessa kohta ja alasveto-valikosta työstettävän dokumentin alta oikea otsikko, jonka alle työkappaleet siirretään.

Kaikkien otsikoiden alle ei välttämättä löydy valmista ladattavaa sisältöä, jolloin sisältö pitää kirjoittaa käsin.



*Kuva 26. Siirto komennon valintaikkuna*

## 4.6 Dokumentaatiohaasteiden ylittäminen

Dokumentaation ja ketterän kehityksen keskeisiä piirteitä tutkiskellessa huomataan, että dokumentaatio keskittyy koko projektiin, mutta ketterä kehittäminen keskittyy projektin suorittamiseen. Tästä johtuen toimivan dokumentaation toteuttaminen ketterässä kehityksessä ei vaadi ketterän kehityksen keskeisistä ajatusmalleista luopumista. Projektin vaatimukset ja tavoitteet sekä aikataulusovitus sovitaan ja dokumentoidaan ennen projektin toteutusta. Ohjelmistokehityksestä vastaava luo vaatimusten ja aikataulutuksen pohjalta suunnittelun, jossa määritellään iteraatioiden tavoitteet siten, että vaatimukset ja aikataulu toteutuisivat sopimuksen mukaisesti.

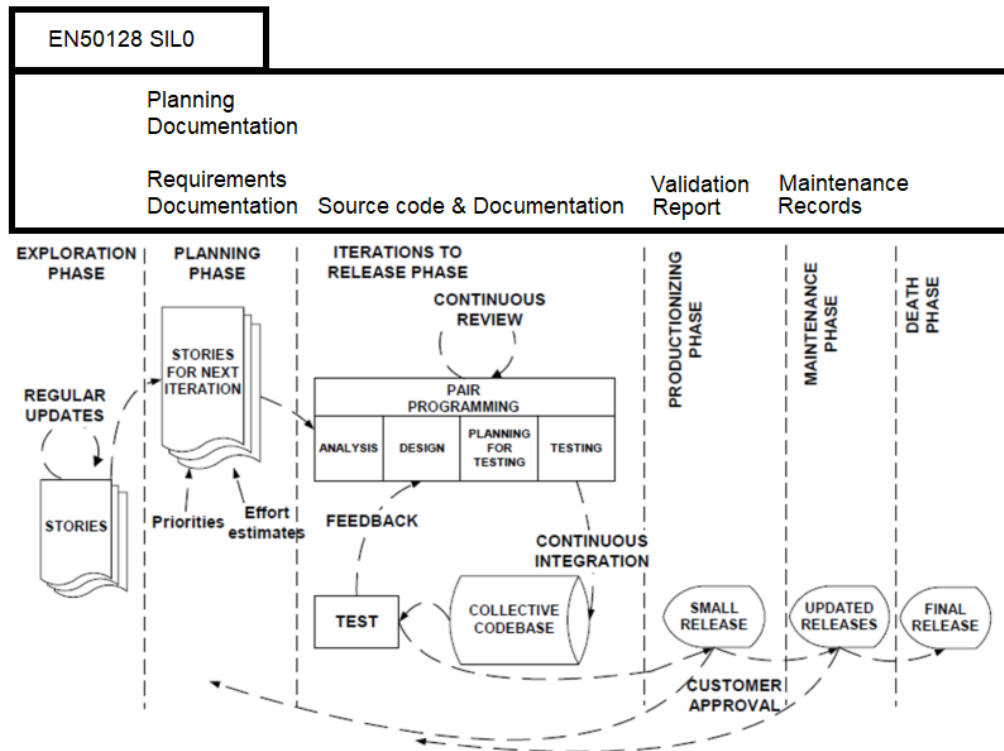
Projektidokumentaatioon sisältyy sovitut aikataulut ja vaatimukset sekä suunnittelusta selkeä tiivistelmä, mutta projektidokumentaation tärkein osa on iteraatioiden eli toteutuksen dokumentointi. Iteraatioiden jälkeen ohjelmistokehittäjät luovat lyhyen yhteenvedon siitä mihin tehtiin muutoksia, minkälaisia ja miksi muutoksia tehtiin. Esimerkiksi: ”Connector komponenttiin muutettiin kohdeosoitteen pingaus protokollaa. Nykyinen protokolla yrittää yhdellä ping-komennolla lähettää viisi pakettia ja komentoa ajetaan kahden sekunnin viiveellä neljä kertaa. Muutos tehtiin, koska edellinen versio jäi ikuisen ping-looppiin jos yhteyttä ei pystytty muodostamaan. Alla linkki uuteen komponentti versioon Connector\_1.0.2”. Iteraatioiden välinen kokous pidetään suullisena tilannekatsauksena, jossa kerrotaan tarkemmin komponentin ominaisuuksista. Jos iteraatiokokouksessa nousee jotain erikoista ja tärkeää esille, ohjelmistokehityksestä vastaava pyytää huomion tehnyttä lisäämään tämän huomion sen iteraation dokumentaatioon.

Dokumentaatiolle tulee olla keskitetty paikka, ketteryyden ylläpitämiseksi. Iteraatiidokumentaatio on helppo ja nopea tehdä wikiin ja wikin hakutoiminto helpottaa tiedon hyödyntämistä projektin jälkeen. Erillisten Word-tiedostojen ongelmana on tiedon pirstaloituminen eri tiedostoihin ja tiedostojen jakautuminen eri kansioihin.

Asiakkaan vaatimat dokumentit ovat projektin vaatimuksia, jotka toteutetaan kuten muutkin projektin vaatimukset. Suunnitteluvaiheessa määritetään, kuka tekee minkäkin asiakasdokumentin ja missä iteraatiossa.

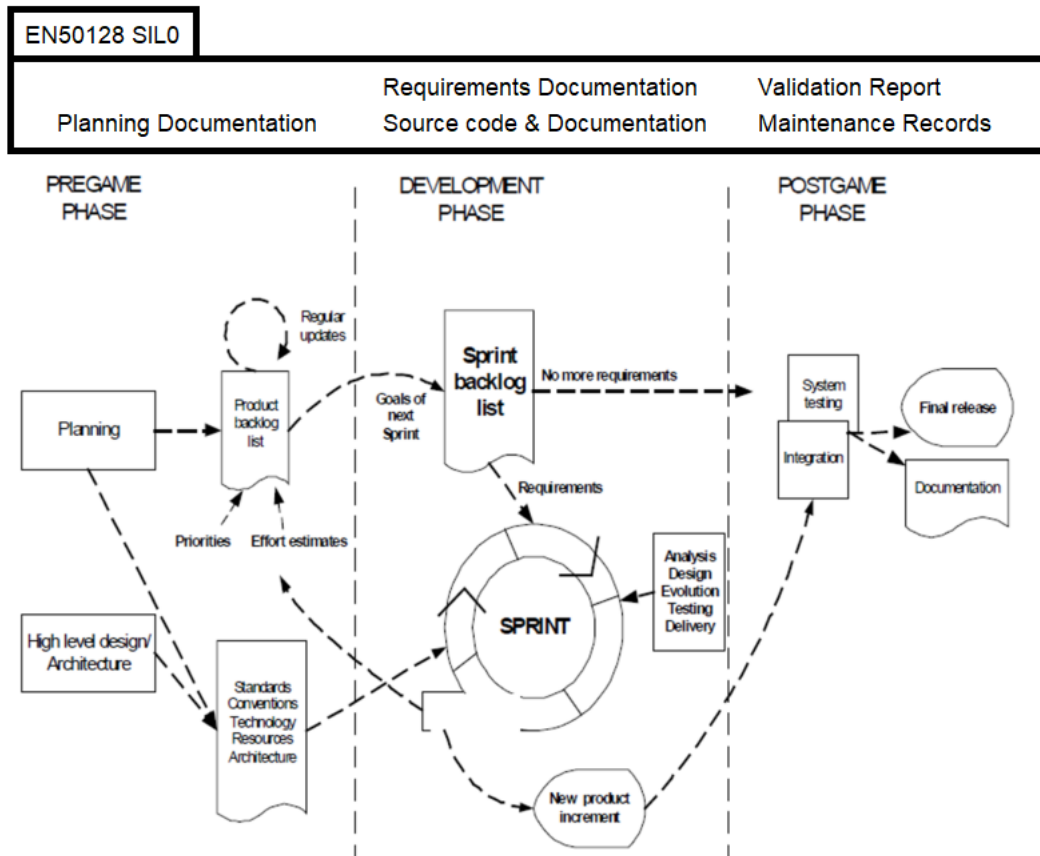
Dokumentaatio ei saa jarruttaa aktiivista kehittämistä vaan sen pitäisi olla luonnollinen osa toimintaa, joka luo perustusta jatkotyölle. Dokumentaatio tulee olla hyvin jaettu ohjelmistokehitysprojektin eri työntekijöille. Esimerkiksi projektipäällikkö tekee vaatimusmäärittelyn/spesifikaatiot, johtava ohjelmistokehittäjä ja projektipäällikkö luo komponenttijaon ja yksittäiset ohjelmistokehittäjät dokumentoivat oman komponenttinsa. Projektin päättyttyä projektipäällikkö ja johtava ohjelmistokehittäjä keräävät kaikki komponenttidokumentaatiot projektituloksiin, tuottaen dokumentaation valmiista tuotteesta.

Rautatiestandardi EN 50128 suosittelee SILO tason dokumentaatioon kuuluviksi: suunnitteludokumentaatiota (engl. Planning documentation), vaatimusdokumentaatiota (engl. Requirements documentation), lähdekoodin ja sen dokumentaation (engl. Source code & documentation), validointiraportin (engl. Validation report) ja ylläpitorekisterin (engl. Maintenance records). Näiden dokumenttien soveltamiseksi ketterän kehityksen tuotantomalliin otettiin esimerkkitapauksiksi extreme programming (XP) ja Scrum (Kuvat 27, 28).



**Kuva 27.** XP-malli ja SILO dokumentaatio vaatimukset, muokattu lähteestä [5]

Kuvassa 27 esitetään XP:n osalta missä vaiheissa vaaditut dokumentit luodaan. XP:n suunnitteluvaiheessa luodaan suunnittelu- ja vaatimusdokumentaatio. Iteraatio ja julkaisuvaiheessa luodaan lähdekoodi ja siihen liittyvä dokumentaatio. Tuotteistusvaiheessa luodaan validointiraportti ja ylläpitovaiheessa ylläpitorekisteri.



*Kuva 28. Scrum malli ja SILO dokumentaatio vaatimukset, muokattu lähteestä [5]*

Kuvassa 28 esitetään Scrum:n osalta missä vaiheissa vaaditut dokumentit luodaan. Scrum:n alkuvaiheessa luodaan projektin suunnittelu, minkä seurauksena suunnitteludokumentaatio tulee samalla luotua. Tuotantovaiheessa luodaan Sprint backlog, johon kirjattuja vaatimuksia käytetään iteraatioiden toteuttamisessa. Näistä vaatimuksista luodaan vaatimusdokumentaatio. Iteraatioiden aikana luodaan lähdekoodi sekä siihen liittyvä dokumentaatio. Jälkipelivaiheessa tehdään järjestelmän testaus ja tuotteen julkaisu. Tässä vaiheessa tehdään validointiraportti sekä ylläpitorekisteri.

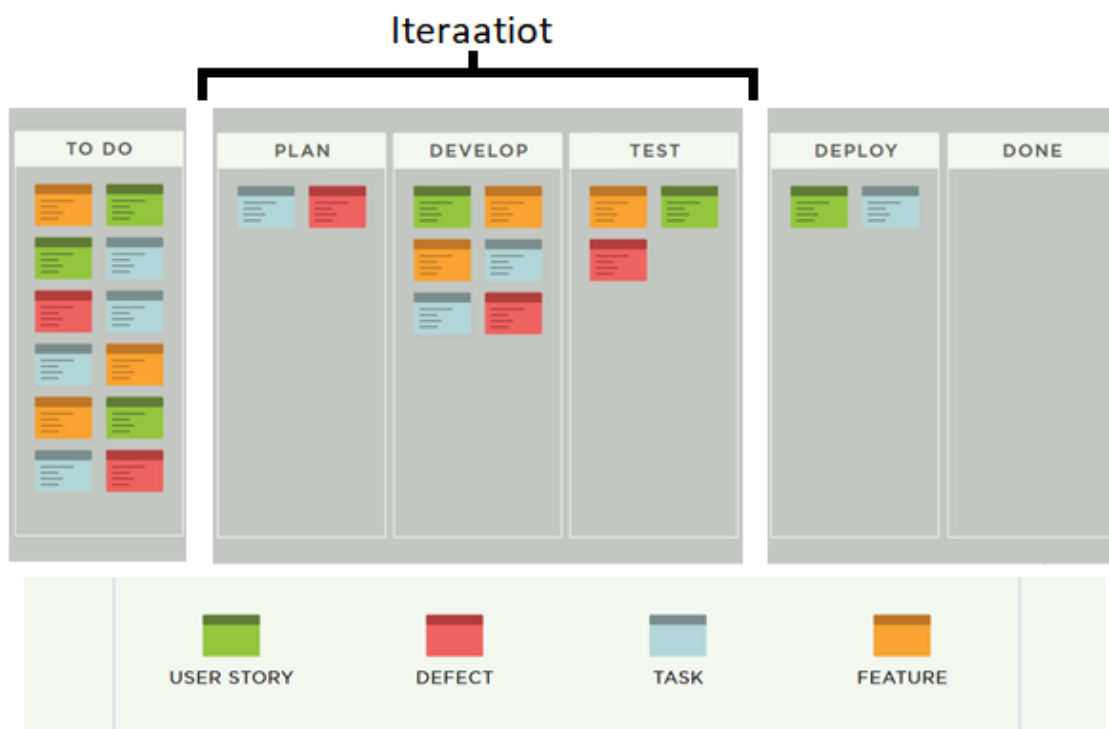
## 4.7 Ketterän dokumentaation toteuttaminen yrityksessä

Yrityksen ei välttämättä tarvitse tuottaa erillistä dokumentaatiota saavuttaakseen rautatiestandardien SILO tason, mutta standardin suositukseksi on, että SILO tasolla luodaan Software Planning Documents, Software Requirements Documents, Source Code & Documentation, Software Validation Report ja Software Maintenance Records. Mikäli SILO tason dokumentaatiota käytetään markkinoinnin välineenä tai yrityksen yleisenä lähtökohtana tuotannossa, tulisi nämä SILO tasolla suositellut dokumentit luoda, jotta näillä väitteillä olisi konkreettista taustaa.

Ketterässä ohjelmistokehityksessä dokumentaatiovaikeudet keskittyvät varsinaiseen iteraatio vaiheeseen, jolloin pääasiassa keskitytään tuotteen toteuttamiseen. Dokumentaation luominen painottuu iteraatioiden välisiin kokouksiin, joissa käydään läpi mitä on tehty, kuinka on edetty, mitä pitää tehdä ja mitä yllätyksiä havaittiin. Kokousta johtavan henkilön tulisi tässä vaiheessa olla aktiivinen ja varmistaa kysymällä, että tärkeät asiat ovat dokumentoitu ylös, jotta vastaaviin yllätyksiin osataan tulevaisuudessa varautua ja että projektin aikana tulleet muutokset ovat ylös kirjattuina.

## 4.8 Ketterien menetelmien tehostaminen

Kanban ja Lean avustavat ketterien menetelmien käyttöä ja helpottaa työn seuranta. Ohjelmistokehityksessä Kanban-taulua voi käyttää sellaisenaan tai Kanban-taulun voi tehdä jokaiselle iteraatiolle. Esimerkiksi kymmenen iteraation ohjelmistokehityksessä luodaan kymmenen iteraatiotaulua, jotka sisältävät suunnittelu, kehitys ja testaus sarakkeet (Kuva 29). Iteraatiokokous vastaa suunnittelu saraketta. Tässä vaiheessa täytetään alkavan iteraation tuotanto- ja testaus-tavoitteet. Seuraavan iteraation alussa suunnitteluvaiheessa eli iteraatiokokouksessa katsotaan, saatiinko tuotanto ja testaus tavoitteet toteutettua. Jos tavoitteita jäi toteuttamatta, niitä voidaan siirtää alkavalle viikolle tai muihin tuleviin iteraatioihin.



*Kuva 29. Kanban ketterässä ohjelmistokehityksessä, muokattu lähteestä [15]*

Lean:in JIT ja Jidoka osat eivät ole ohjelmistokehityksen näkökulmasta hyödyllisiä. Kaizen yritysfilosofia ja sen pohjalta kehitetty TPS sisältävät kaikenlaista työntekoa hyödyttäviä näkökulmia, johtotapoja sekä työkaluja. Osa näistä ovat itsestään selvyksiä, mutta niitä on hyvä ajatella yrityksen näkökulmasta ja varmistaa, että niiden mukaan yrityksessä toimitaan. Tämän tyyppisiä ovat kaikkia kohtaan tasapuolinen kohtelu, luottamus ja yhteistyö. Työtä tehdessä tulee pitää mielessä Kaizen, jossa pyritään parantamaan yrityksen prosesseja pienillä, mutta tehokkailla tavoilla. Monesti parannukset syntyvät, kun työtä tehdessään työntekijät törmäävät hankaluuksiin tai turhiin toimenpiteisiin ja kertovat näistä esimiehilleen. Lean:ssä keskitytään myös poistamaan ylimääräiset asiakkaan toiveista. Tämä tarkoittaa sitä, että omalla asiantuntemuksella pyritään neuvomaan asiakasta, kun sovitaan ohjelmiston vaatimuksista ja toiminnallisuudesta.

## 5. TULOSTEN ARVIOINTI

Tässä kappaleessa arvioidaan työssä aikaansaatuja tuloksia ja verrataan niitä tutkimustavoitteisiin. Työssä tutkittiin yrityksen dokumentaation tilaa sekä tarpeita. Tutkimuskysymyksiä olivat:

- Dokumentaation tarve ja hyödyt ketterässä kehityksessä?
- Onko dokumentaatio siihen käytetyn ajan arvoista?
- Mitä on rautatiestandardien mukainen dokumentaatio ketterässä ohjelmistokehityksessä?
- Kuinka rautatiestandardien mukainen dokumentaatio ja ketterä ohjelmistokehitys voidaan toteuttaa kohdeyrityksessä?

### 5.1 Dokumentaation tarve ja hyödyt ketterässä kehityksessä

Dokumentaatio on keskeinen osa kommunikaatiota niin työntekijöiden kuin yritysten välillä. Sen suurimpana erona suulliseen tiedonviestintään on säilyvyys. Lyhyellä tähtämellä suullinen kommunikaatio on sujuvampaa, mutta pitkällä tähtämellä se tarkoittaa, että samat asiat pitää keksiä ja kertoa yhä uudestaan ja uudestaan. Tämän takia on tärkeää löytää kehityksen ja jatkuvuuden kannalta tärkeimmät asiat, jotka kirjataan ylös. Dokumentaation tarkoitus ketterässä kehityksessä on taata, että seuraava työ sujuu paremmin ja seuraavalle innovaatiolle on tietopohja, josta rakentaa. Näitä käsiteltiin luvuissa 2.1 ja 2.5

### 5.2 Dokumentaation arvo

Dokumentaation arvo perustuu sen antamaan tulevaisuuden hyötyyn ja siihen käytettyyn aikaan. Vesiputousmalliin nähden ketterässä ohjelmistokehityksessä on ajatus siitä, että dokumentaatio voi määrällisesti olla kevyempi ja monesti on tarve muutoksille. Kappaleessa 3.2 käsiteltiin työssä tehtyä haastattelua, jonka tuloksena 66% haastateltavista koki, että dokumentaatioon tulisi käyttää nykyistä enemmän aikaa. Taulukosta 6 havaitaan, että apua koodin ymmärtämiseen yksi etsii wiki-sivulta, neljä koodin kommentteista ja kaksi kysyy työkaverilta. Tämän lisäksi neljä vastasi dokumentoivansa koodiin, yksi wiki-sivuille ja kaksi virallisiin dokumentteihin. Haastattelusta ilmeni, että dokumentaatiota tallennetaan useaan eri paikkaan ja monesti dokumentaatioissa on puutteita. Tästä syystä dokumentaation hyöty menetetään tiedon etsintään kulutettavalla ajalla. Haastattelun pohjalta tehdyt korjaukset dokumentaatioon kasvattavat dokumentaation arvoa. Työ keskittyi yrityksen ohjelmistodokumentaatioon, minkä takia työssä ei pystytty käsittelemään projektidokumentaation tasoa eri osastojen välillä ja niiden vaikutuksia. Tämän tyyppisiä vaikutuksia ovat esimerkiksi: asiakkaalle myydyin järjestelmän ja olemassa olevien järjestelmien eroavaisuuksien vaikutukset sekä ohjelmistokehitykseen, että tuotantoon; uusien ohjelmisto-ominaisuuksien luomat laitevaatimukset. Kun dokumentaatio on saatu yhteen

paikkaan ja dokumentaation luomisesta on yhteneväinen yrityskulttuuri, voidaan tulevaisuudessa tutkia voiko jotain dokumentaatiota jättää tekemättä ja tällä tavoin keventää työtaakkaa.

### 5.3 Rautatiestandardien mukainen dokumentaatio ketterässä ohjelmistokehityksessä

Työssä tutkittiin standardin asettamia vaatimuksia ohjelmistodokumentaatiolle. Vaatimusten pohjalta luotiin dokumentaatio pohjat (Liite B, C, D, E) ja vaadittuja dokumentteja sovitettiin ketterien menetelmien runkoihin.

Polarion:ia hyödyntäen yritysdokumentaatiota muutettiin standardin mukaiseksi. Kappaleessa 4.6 todetaan, että dokumentaation ja ketterän kehityksen keskeisiä piirteitä tutkiskellessa huomataan, että dokumentaatio keskittyy koko projektiin, mutta ketterä kehittäminen keskittyy projektin suorittamiseen. Tästä johtuen toimivan dokumentaation toteuttaminen ketterässä kehityksessä ei vaadi ketterän kehityksen keskeisistä ajatusmalleista luopumista.

Kuvassa Kuva 27 esitetään XP:n osalta missä vaiheissa vaaditut dokumentit luodaan. XP:n suunnitteluvaiheessa luodaan suunnittelu- ja vaatimuskumentaatio. Iteraatio- ja julkaisuvaiheessa luodaan lähdekoodi ja siihen liittyvä dokumentaatio. Tuotteistusvaiheessa luodaan validointiraportti ja ylläpitovaiheessa ylläpitorekisteri.

Kuvassa Kuva 28 esitetään Scrum:n osalta missä vaiheissa vaaditut dokumentit luodaan. Scrum:n alkuvaiheessa luodaan projektin suunnittelu, minkä seurauksena suunnitteludokumentaatio tulee samalla luotua. Tuotantovaiheessa luodaan Sprint backlog, johon kirjattuja vaatimuksia käytetään iteraatioiden toteuttamisessa. Näistä vaatimuksista luodaan vaatimuskumentaatio. Iteraatioiden aikana luodaan lähdekoodi sekä siihen liittyvä dokumentaatio. Jälkivaiheessa tehdään järjestelmän testaus ja tuotteen julkaisu. Tässä vaiheessa tehdään validointiraportti sekä ylläpitorekisteri.

Kanban:ia voidaan hyödyntää ketterien menetelmien tehostamiseksi. Kappaleen 5.8 kuvassa Kuva 29 esitellään kuinka perinteisen Kanban rakenne paloitellaan ketterämmäksi. To do osio käsittää projektin suunnittelu vaiheen. Plan -vaiheessa suunnitellaan iteraatioissa tehtävät osa-alueet. Develop -vaiheessa ohjelmistoa kehitetään ja test -vaiheessa testataan. Plan, develop ja test muodostavat iteraatiot, joiden jälkeen ohjelmisto otetaan käyttöön ja projekti päättyy. Jatkuvan käyttöönoton mallissa deploy -vaihe liitetään iteraatioihin.

Vanhassa toimintamallissa ohjelmistokehitys oli ketterää, mutta dokumentaatio vähäistä eikä se tukenut ulkoisen dokumentaation tarpeita. Tästä johtuen ohjelmistokehitysprosessin jälkeen alkoi standardien ja asiakkaan vaatimien dokumenttien luonti. Nämä dokumentit sisältävät tietoa ohjelmistokehitysprosessista. Vähäisen ohjelmistodokumentaation takia jälkikäteen tarkan dokumentaation luonti on erittäin työlästä tutkimustyötä. Työssä luodussa mallissa nämä dokumentit luodaan ohjelmistokehityksen aikana, jolloin ylös kirjattavat asiat tulevat projektille luontaisesti vastaan. Tällöin säästetään paljon dokumentaation luomiseen käytettävää tutkimusaikaa ja dokumentoidut asiat ovat toteutushetken varmaa tietoa eivätkä tutkimuksen ja muistin hämärtämiä. Työssä esitettyjen dokumentaatiota

helpottavien ohjelmistojen käyttö keskittää dokumentaation yhteen paikkaan, jolloin haastatteluissa esille noussut tiedon pirstaloituminen vähenee. Näiden ohjelmistojen avulla voidaan hyödyntää työssä luotuja dokumentaatio pohjia. Tietoa voidaan kopioida ja liittää suoraan otsikoiden alle.

Tulevaisuudessa tutkimuksena olisi tärkeää uudistaa rautatiestandardit käsittelemään myös ketterää ohjelmistokehitystä. On myös tarve ketteränkehityksen mallien tarkempaan jaotteluun. Monet kirjat kuvailevat erilaisia ketteränkehityksen malleja samoilla keskeisillä periaatteilla ja avain konsepteilla. Yleisellä tasolla ketteränkehityksen keskeiset periaatteet ovat samoja, minkä takia erilaisissa ketterissä menetelmissä pitäisi keskittyä siihen mikä tekee mallista erityisen. Työssä törmäsi monesti tilanteisiin, joissa huomasi että jokaisella on oma käsitys mitä on ketteräkehitys.

## 6. YHTEENVETO

Perinteisen vesiputousmallin käyttö nykyajan ohjelmistokehityksessä ei ole käytännöllistä, koska se ei mahdollista muutoksien luomista eikä alussa tehtyjen virheiden korjaamista. Toisaalta ketterät menetelmät, jotka pyrkivät vähäiseen tai koodikeskeiseen dokumentaatioon toimivat parhaiten yrityksissä, jotka tekevät vain ohjelmistoja. Jos ohjelmointi on vain osayritystä, ei ketterien menetelmien dokumentaatio hyödytä yrityksen muita osastoja. DevOps ei ole Mitron Oy:lle tarpeellinen ratkaisu, koska tuotteita ei Mitron Oy:n puolesta aktiivisesti ylläpidetä ja päivitetä. Laitevalmistuksen ryhmä tekee usein käyttöönoton kehitystiimin avustuksella. DevOps:in käyttämiseksi Mitron Oy:n tulisi perustaa erillinen käyttöönotto ryhmä, jonka pitäisi tehdä yhteistyötä sekä ohjelmistokehitys että laitevalmistus ryhmien kanssa. Käytännössä kaikkea toimintaa tulisi muuttaa, jotta käyttöönotto ryhmä mahtuisi toimintaan mukaan.

Ketterien menetelmien pääasiallinen ero perinteisiin ohjelmistokehitysmalleihin on sprintti/iteraatio -vaihe, jossa iteraatioista vastaavan henkilön pitää huolehtia, että tuotetaan koko organisaatiota hyödyttävää dokumentaatiota.

Rautatiestandardi IEEE 1558 asettaa dokumentaatiotyyppeiden valinnan hankkijalle ja vaatii dokumentteiltaan liitteissä B, C, D, E esiteltyä rakennetta. EN 50128 asettaa SIL0 tasolle vaatimuksiksi ohjelmistosuunnitteludokumentit, ohjelmistovaatimusdokumentit, ohjelmistokoodin ja sen dokumentaation, ohjelmiston validointi raportin ja ohjelmiston ylläpito rekisterit.

Dokumentaatiota tukemaan voi hyödyntää erilaisia ohjelmistoja, kuten Jira ja Polarion. Työssä käsiteltiin Polarion:in kykyä rautatiestandardien mukaisen dokumentaation luomiseen. Dokumentaation rakenne kärsii, kun sen osista tehdään työkappaleita. Jos rautatiestandardin vaatimusten mukaan tehty dokumenttipohja pysyy muuttumattomana, niin dokumentti täyttää rakenteelliset vaatimukset. Ohjelmien käyttö pakottaa työntekijät tekemään dokumentaation yhteen tiettyyn sijaintiin ja samalla formaatilla.

Yrityksen tulisi ensin korjata yritysdocumentaatiokulttuuri, jotta kaikki tekevät saman tyyppistä dokumentaatiota ja dokumentaatio tallennetaan yhteen paikkaan. Tällöin dokumentaatio ei pirstaloidu eri formaattien ja sijaintien mukaan. Kun näihin tavoitteisiin on päästy, voidaan tulevaisuudessa tutkia voiko dokumentaatiomäärää keventää. Yksinkertaisuudessaan liika dokumentaatio syö työaika, mutta liian vähäinen tehdyn työn tehokkuutta. Perusongelmat tulee korjata, jotta yrityksen sisäistä dokumentaatiota voidaan alkaa räätälöidä tehokkaammaksi.

Rautatiestandardien vaatimukset voidaan sulauttaa ketterien menetelmien runkoon, kuten kappaleessa 4.6 sivulla 47 ja 48 on esitetty. Kanban:ia voidaan käyttää ketterien menetelmien tehostamiseen. Kappaleessa 4.8 sivulla 49 on esitelty kuinka Kanban:in rakennetta voidaan muokata ketterän kehityksen vaatimusten mukaan siten, että se huomioi iteraatiot tehokkaammin.

## LÄHTEET

- [1] IEC 62279: Railway applications – Communications, signaling and processing systems – Software for railway control and protection systems (EN 50128) 2001
- [2] EN 50128: Railway applications – Communications, signaling and processing systems – Software for railway control and protection systems 2001
- [3] IEEE 1558 Standard for Software Documentation for Rail Equipment and Systems
- [4] I. Haikala, J. Märijärvi, Ohjelmistotuotanto, Talentum Helsinki 2004, 440 s.
- [5] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta, A.G.I.L.E 2002, VTT Publications 478. Saatavissa: <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf> Tarkistettu: 1.4.2017
- [6] Henrik Kniberg, Kanban vs Scrum, Version 1.1 (2009-06-29), InfoQ. Saatavissa: <https://www.crisp.se/file-uploads/Kanban-vs-Scrum.pdf> Tarkistettu: 16.11.2017
- [7] Ian Sommerville, Software Documentation 2011, revised version of chapter 30 from the book Software Engineering, 4<sup>th</sup> edition published by Addison Wesley 1992. Saatavissa: <http://www.literateprogramming.com/documentation.pdf> Tarkistettu: 22.2.2018
- [8] T. Stober, U. Hansmann, Agile Software Development, Springer-Verlag Berlin Heidelberg 2010. Saatavissa: <https://www.google.fi/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahU-KEWjO5cjKwJnaAhUqP5oKHxpMCAoQFggmMAA&url=http%3A%2F%2Fwww.springer.com%2Fcontent%2Fdocument%2Fcontent%2Fdownloadaddocument%2F9783540708308-c1.pdf%3FSGWID%3D0-0-45-808609-p173832350&usg=AOvVaw1Xioz9ayDQsn-i5kDJRyXS> Tarkistettu: 1.4.2018
- [9] Mike Holcombe, Running an Agile Software Development Project, John Wiley & Sons 2008, 328 s.
- [10] Martin Fowler, Manifesto for Agile Software Development, 2001. Saatavissa: <http://agilemanifesto.org/> Tarkistettu: 31.10.2018
- [11] Martin Fowler, 22 March 2005. Saatavissa: <https://martinfowler.com/bliki/CodeAsDocumentation.html> Tarkistettu: 1.4.2018
- [12] Michael James, Luke Walter, Scrum Reference Card. Saatavissa: [https://www.collab.net/sites/default/files/uploads/CollabNet\\_scrumreferencecard.pdf](https://www.collab.net/sites/default/files/uploads/CollabNet_scrumreferencecard.pdf) Tarkistettu: 28.11.2018

- [13] Kent Beck, Cynthia Andres, Extreme Programming Explained Second Edition 2004, Addison-Wesley, 189 s.
- [14] Sharifah Syed-Abdullah, Mike Holcombe, Marian Gheorghe, Agile Methodology in Practice, Saatavissa: <http://www.dcs.shef.ac.uk/intranet/research/public/resmes/CS0304.pdf> Tarkistettu: 28.11.2018
- [15] KANBAN ROADMAP: How to get started in 5 easy steps, Leankit. Saatavissa: <https://go.leankit.com/kanban-roadmap.html> Tarkistettu: 16.11.2017
- [16] David J Anderson, Andy Carmichael, Essential Kanban Condensed, Lean Kanban University Press 2016, Series: Essential Kanban. Saatavissa: <http://leankanban.com/wp-content/uploads/2016/06/Essential-Kanban-Condensed.pdf> Tarkistettu: 28.11.2018
- [17] Thorsten Ahrens, Lean production: Successful implementation of organisational change in operations instead of short term cost reduction efforts (2006), Lean Alliance. Saatavissa: [http://www.lean-alliance.com/en/images/pdf/la\\_lean\\_survey.pdf](http://www.lean-alliance.com/en/images/pdf/la_lean_survey.pdf) Tarkistettu: 25.3.2018
- [18] Manjunath Shettar, Pavan Hiremath, Nikhil R, Vithal Rao Chauhan, KAIZEN – A case study, Journal of Engineering Research and Applications Vol. 5, Issue 5, Ijera. Saatavissa: <https://pdfs.semanticscholar.org/255e/1af8dff444c8f594675ceb57600b51bf2e72.pdf> Tarkistettu: 28.3.2018
- [19] Xiaofeng Wang, The Combination of Agile and Lean in Software Development: An Experience Report Analysis, Published 2011 in 2011 AGILE Conference, IEEE. Saatavissa: <https://pdfs.semanticscholar.org/c694/ed34f1c56d115f9c1b35f5625a86cee2f7da.pdf> Tarkistettu: 16.11.2017
- [20] Maciej Pienkowski, Waste measurement techniques for lean companies. International Journal of Lean Thinking Volume 5, Issue 1 (December 2014), Teknokent. Saatavissa: [http://thinkinglean.com/img/files/Maciej\\_Piekowski.pdf](http://thinkinglean.com/img/files/Maciej_Piekowski.pdf) Tarkistettu: 25.3.2018
- [21] Sanjeev Sharma, Bernie Coyne. DevOps 3<sup>rd</sup> IBM Limited Edition, John Wiley & Sons 2017. Saatavissa: [https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahU-KEwiTtcTS4cTeAhWKhSwKHVkoDkwQFjAAegQICRAC&url=http%3A%2F%2Fpos-lis.fon.bg.ac.rs%2Findex.php%3Foption%3Dcom\\_docman%26task%3Ddoc\\_download%26gid%3D667%26Itemid%3D26&usg=AOvVaw1IsG40owCawezHAoBbrqiR](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahU-KEwiTtcTS4cTeAhWKhSwKHVkoDkwQFjAAegQICRAC&url=http%3A%2F%2Fpos-lis.fon.bg.ac.rs%2Findex.php%3Foption%3Dcom_docman%26task%3Ddoc_download%26gid%3D667%26Itemid%3D26&usg=AOvVaw1IsG40owCawezHAoBbrqiR) Tarkistettu: 8.11.2018
- [22] Simon Foster, 27 September 2014. Saatavissa: <https://medium.com/diaries-of-it/what-is-the-difference-between-development-and-operations-eb0db9c7733e> Tarkistettu: 28.11.2018
- [23] Karthiga Sadasivan, DevOps, Happiest minds 2014, 10 s.

- [24] James Lee, CA technologies, A Pragmatic Guide to Getting Started with DevOps, LevelUP, 35 s.
- [25] Gene Kim, Jez Humble, Patrick Debois, John Willis, The DevOps Handbook 2016, IT Revolution Press. Saatavissa: [http://images.itrevolution.com/documents/DevOps\\_Handbook\\_Intro\\_Part1\\_Part2.pdf](http://images.itrevolution.com/documents/DevOps_Handbook_Intro_Part1_Part2.pdf) Tarkistettu: 28.11.2018
- [26] DevOps Institute, DevOps The Basics. Saatavissa: <https://www.traincanada.com/files/downloads/DevOps-The-Basics.pdf>

## **LIITE A: HAASTATTELU LOMAKE**

### **HAASTATTELU**

#### **Yleistä**

Kuinka paljon aikaa dokumentaatio vie suhteessa tuottavaan aikaan?

Mitä mieltä olet siitä jos sinun pitäisi käyttää enemmän työaikaa dokumentointiin?

Onko yleisesti ottaen projektin edetessä jokin vaihe sellainen missä teidän mielestänne kuluu liikaa aikaa? On kuin projektin pullonkaula?

Mikä on olennaista käsitellä tässä työssä?

### **Sales**

Mitä mieltä asiakkaat ovat mitronin dokumentaatiosta? Tarviiko aiheesta aina sopia erikseen? Onko suppea / laaja?

Minkälaisia vaatimuksia asiakkaat asettavat dokumentaatiolle? (Kuinka nopeasti?)

Päästäänkö näihin tavoitteisiin?

Vaikuttaako Mitronin dokumentaatio asiakaskuntaan?

#Minkä tason dokumentaatio on myyjille hyödyllistä?

Voiko dokumentaatio vaikuttaa markkinointiin? Järjestelmä / Platform dokumentaatio?

## **SW**

Mikä on ohjelmisto tuotannon kannalta pullonkaula?

Mikä on ongelmallisinta ohjelmisto projekteissa?

Millä tavalla vanhaan tai jonkun muun tekemään koodiin suhtaudutaan?

Mistä haet apua/dokumentaatiota vanhan koodin ymmärtämiseen? Koodiin kommentoitu / erillinen dokumentti / google?

Missä vaiheessa projektia teet dokumentointia? Minkälaista SW (koodiin kommentointi / muu dokumentti?), testaus, yleinen? Miksi näissä vaiheissa?

Törmäätkö usein tilanteeseen, ettet tiedä miten jokin toiminnallisuus on koodattu etkä löydä mistään ohjeita?

## **Project**

Kuinka projektit suunnitellaan?

Onko tilanteita joissa ihmetellään, kuinka tähän tilanteeseen päädyttiin?

Onko projektien jäljitettävyys tärkeää?

Onko yleisellä tasolla projekteissa joitain tärkeitä tilanteita/kohtia joissa pitää dokumentoida?  
Alku/koodaus/testaus/loppu?

Minkälainen dokumentaatio on tärkeintä?

Muuta lisättävää?

# LIITE B: SRS POHJA

**MITRON**

		Date	29.11.2016
		Pages	1 / 12
Author	<u>EXAMPLE: T. Wilson</u> (SW Developer)	Review	<u>EXAMPLE: A. Holm</u> (SW Lead)
Revision	EXAMPLE: 1.2	Approval	<u>EXAMPLE: R. Hakola</u> (Project Manager)
Document description	Project name Software Requirements Specification		

## Project name Software Requirements Specification

Version	Description	Author	Date
1	Initial revision.	<u>T. Wilson</u>	08.06.2015
1.1	EXAMPLE: Removed features concerning Free Hand service	<u>T. Wilson</u>	03.10.2016
1.2	EXAMPLE: Added IEEE reference and functional requirement ID's	<u>A. Ketola</u>	29.11.2016

MITRON

		<b>Date</b>	29.11.2016
		<b>Pages</b>	2 / 12
<b>Author</b>	EXAMPLE: T. Wilson (SW Developer)	<b>Review</b>	EXAMPLE: A. Holm (SW Lead)
<b>Revision</b>	EXAMPLE: 1.2	<b>Approval</b>	EXAMPLE: R. Hakola (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		

## REVISION SHEET

Revision number	Revision date	Change description

MITRON

		Date	29.11.2016
		Pages	3 / 12
Author	EXAMPLE:T.Wilson (SW Developer)	Review	EXAMPLE:A.Holm (SW Lead)
Revision	EXAMPLE:1.2	Approval	EXAMPLE:R.Hakola (Project Manager)
Document description	Project name Software Requirements Specification		

## CONTENTS

<b>1 Introduction</b>	<b>6</b>
1.1 Purpose	6
1.2 Scope	6
1.3 Definitions, acronyms, and abbreviations	6
1.4 References	6
1.5 Overview	7
1.6 Maintenance	7
<b>2 Overall description</b>	<b>7</b>
2.1 Product perspective	7
2.1.1 System interfaces	7
2.1.2 User interfaces	7
2.1.3 Hardware interfaces	7
2.1.4 Software interfaces	7
2.1.5 Communication interfaces	8
2.1.6 Memory constraints	8
2.1.7 User operations	8
2.1.8 Site adaptation requirements	8
2.2 Product functions	8
2.3 User characteristics	8
2.4 Constraints	8
2.5 Assumptions and dependencies	8
2.6 Apportioning of requirements	8
<b>3 Specific requirements</b>	<b>8</b>
3.1 External interfaces	8
3.2 Features	9
3.2.1 Feature 1	9
3.2.1.1 Introduction/Purpose of feature	9
3.2.1.2 Stimulus/Response sequence	9
3.2.1.3 Associated functional requirements	9
3.2.1.3.1 Functional requirement 1	9
3.2.1.3.2 Functional requirement 2	9
3.2.1.3.3 Functional requirement 3	9
3.2.2 Feature 2	9
3.2.2.1 Introduction/Purpose of feature	9
3.2.2.2 Stimulus/Response sequence	9
3.2.2.3 Associated functional requirements	10
3.2.2.3.1 Functional requirement 1	10
3.2.2.3.2 Functional requirement 2	10
3.2.2.3.3 Functional requirement 3	10
3.2.3 Feature 3	10
3.2.3.1 Introduction/Purpose of feature	10
3.2.3.2 Stimulus/Response sequence	10
3.2.3.3 Associated functional requirements	10
3.2.3.3.1 Functional requirement 1	10

MITRON

		<b>Date</b>	29.11.2016
		<b>Pages</b>	4 / 12
<b>Author</b>	<del>EXAMPLE:T.Wilson</del> (SW Developer)	<b>Review</b>	<del>EXAMPLE:A.Holm</del> (SW Lead)
<b>Revision</b>	EXAMPLE:1.2	<b>Approval</b>	<del>EXAMPLE:P.Hakola</del> (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		

3.2.3.3.2	Functional requirement 2 .....	10
3.3	Performance requirements .....	11
3.4	Logical database requirements .....	11
3.5	Design constraints .....	11
3.5.1	Standards compliance .....	11
3.5.2	Hardware limitations .....	11
3.6	Software system attributes .....	11
3.6.1	Reliability .....	11
3.6.2	Availability .....	11
3.6.3	Security .....	11
3.6.4	Maintainability .....	11
3.6.5	Portability .....	11

**MITRON**

		<b>Date</b>	29.11.2016
		<b>Pages</b>	5 / 12
<b>Author</b>	EXAMPLE:T.Wilson (SW Developer)	<b>Review</b>	EXAMPLE:A.Holm (SW Lead)
<b>Revision</b>	EXAMPLE:1.2	<b>Approval</b>	EXAMPLE:R.Hakola (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		

## LIST OF FIGURES

**Kuvaotsikkoluettelon hakusanoja ei löytynyt.**

MITRON

		<b>Date</b>	29.11.2016
		<b>Pages</b>	6 / 12
<b>Author</b>	EXAMPLE.T.Wilson (SW Developer)	<b>Review</b>	EXAMPLE.A.Holm (SW Lead)
<b>Revision</b>	EXAMPLE:1.2	<b>Approval</b>	EXAMPLE.R.Hakola (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		

## LIST OF TABLES

Table 1: Definitions, acronyms, and abbreviations .....	8
Table 2: References.....	9
Table 3: System interfaces .....	9
Table 4: Hardware interfaces.....	9
Table 5: Software interfaces .....	10
Table 6: Communication interfaces .....	10

MITRON

		Date	29.11.2016
		Pages	7 / 12
Author	EXAMPLE:T.Wilson (SW Developer)	Review	EXAMPLE:E.Holm (SW Lead)
Revision	EXAMPLE:1.2	Approval	EXAMPLE:B.Hakola (Project Manager)
Document description	Project name Software Requirements Specification		

# 1 Introduction

## 1.1 Purpose

## 1.2 Scope

## 1.3 Definitions, acronyms, and abbreviations

Term used	Meaning	Note
EA	Example Abbreviation	Customer

**Table 1: Definitions, acronyms, and abbreviations**

## 1.4 References

Reference	Revision	Title
/1/	Draft17	EXAMPLE PROJECT Functional and Technical Description
/2/	1.0	EXAMPLE RFC 3416 – Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)

**MITRON**

		<b>Date</b>	29.11.2016
		<b>Pages</b>	8 / 12
<b>Author</b>	EXAMPLE:T.Wilson (SW Developer)	<b>Review</b>	EXAMPLE:A.Holm (SW Lead)
<b>Revision</b>	EXAMPLE:1.2	<b>Approval</b>	EXAMPLE:R.Hakola (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		


**Table 2: References**

## 1.5 Overview

## 1.6 Maintenance

# 2 Overall description

## 2.1 Product perspective

### 2.1.1 System interfaces

System interface	Protocol	Description

**Table 3: System interfaces**

### 2.1.2 User interfaces

### 2.1.3 Hardware interfaces

Interface	Protocol	Description

**Table 4: Hardware interfaces**

MITRON

		Date	29.11.2016
		Pages	9 / 12
Author	EXAMPLE:T.Wilson (SW Developer)	Review	EXAMPLE:A.Holm (SW Lead)
Revision	EXAMPLE:1.2	Approval	EXAMPLE:R.Hakola (Project Manager)
Document description	Project name Software Requirements Specification		

#### 2.1.4 Software interfaces

Interface	Protocol	Description

**Table 5: Software interfaces**

#### 2.1.5 Communication interfaces

Interface	Version number	Description	Reference

**Table 6: Communication interfaces**

#### 2.1.6 Memory constraints

#### 2.1.7 User operations

#### 2.1.8 Site adaptation requirements

### 2.2 Product functions

### 2.3 User characteristics

### 2.4 Constraints

### 2.5 Assumptions and dependencies

### 2.6 Apportioning of requirements

MITRON

		Date	29.11.2016
		Pages	10 / 12
Author	EXAMPLE.T.Wilson (SW Developer)	Review	EXAMPLE.A.Holm (SW Lead)
Revision	EXAMPLE:1.2	Approval	EXAMPLE.R.Hakola (Project Manager)
Document description	Project name Software Requirements Specification		

### 3 Specific requirements

#### 3.1 External interfaces

#### 3.2 Features

##### 3.2.1 Feature 1

###### 3.2.1.1 Introduction/Purpose of feature

###### 3.2.1.2 Stimulus/Response sequence

- 

**Stimulus:**

**Response:**

- 

**Stimulus:**

**Response:**

###### 3.2.1.3 Associated functional requirements

###### 3.2.1.3.1 Functional requirement 1

ID:

###### 3.2.1.3.2 Functional requirement 2

ID:

###### 3.2.1.3.3 Functional requirement 3

ID:

##### 3.2.2 Feature 2

###### 3.2.2.1 Introduction/Purpose of feature

###### 3.2.2.2 Stimulus/Response sequence

-

MITRON

		Date	29.11.2016
		Pages	11 / 12
Author	EXAMPLE:T.Wilson (SW Developer)	Review	EXAMPLE:A.Holm (SW Lead)
Revision	EXAMPLE:1.2	Approval	EXAMPLE:R.Hakola (Project Manager)
Document description	Project name Software Requirements Specification		

**Stimulus:**

**Response:**

- 

**Stimulus:**

**Response:**

### 3.2.2.3 Associated functional requirements

#### 3.2.2.3.1 Functional requirement 1

ID:

#### 3.2.2.3.2 Functional requirement 2

ID:

#### 3.2.2.3.3 Functional requirement 3

ID:

### 3.2.3 Feature 3

#### 3.2.3.1 Introduction/Purpose of feature

#### 3.2.3.2 Stimulus/Response sequence

- 

**Stimulus:**

**Response:**

- 

**Stimulus:**

**Response:**

#### 3.2.3.3 Associated functional requirements

##### 3.2.3.3.1 Functional requirement 1

ID:

**MITRON**

		<b>Date</b>	29.11.2016
		<b>Pages</b>	12 / 12
<b>Author</b>	EXAMPLE.T.Wilson (SW Developer)	<b>Review</b>	EXAMPLE.A.Holm (SW Lead)
<b>Revision</b>	EXAMPLE:1.2	<b>Approval</b>	EXAMPLE.R.Hakola (Project Manager)
<b>Document description</b>	Project name Software Requirements Specification		

### 3.2.3.3.2 Functional requirement 2

ID:

## 3.3 Performance requirements

### 3.4 Logical database requirements

## 3.5 Design constraints

### 3.5.1 Standards compliance

### 3.5.2 Hardware limitations

## 3.6 Software system attributes

### 3.6.1 Reliability

### 3.6.2 Availability

### 3.6.3 Security

### 3.6.4 Maintainability

### 3.6.5 Portability

# LIITE C: SDD POHJA

**MITRON**

		Date	26.05.2014
		Pages	1 / 6
Author		Review	
Revision		Approval	
Document description	Project name Software Design Description		

Project name  
Software Design Description

Version	Description	Author	Date

**MITRON**

		Date	26.05.2014
		Pages	2 / 6
Author		Review	
Revision		Approval	
Document description	Project name Software Design Description		

## REVISION SHEET

Revision number	Revision date	Change description

MITRON

		<b>Date</b>	26.05.2014
		<b>Pages</b>	3 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Design Description		

## CONTENTS

<b>1</b>	<b>Introduction</b> .....	<b>3</b>
1.1	Purpose .....	3
1.2	Scope .....	3
1.3	Definitions, acronyms, and abbreviations .....	3
1.4	Document organization.....	3
<b>2</b>	<b>References</b> .....	<b>3</b>
<b>3</b>	<b>Software design</b> .....	<b>3</b>

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	4 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Design Description		

## LIST OF FIGURES

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	5 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Design Description		

## LIST OF TABLES

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	6 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Design Description		

## 1 Introduction

### 1.1 Purpose

### 1.2 Scope

### 1.3 Definitions, acronyms, and abbreviations

### 1.4 Document organization

## 2 References

## 3 Software design

# LIITE D: SRTM POHJA

**MITRON**

	Date	26.05.2014
	Pages	1 / 6
Author	Review	
Revision	Approval	
Document description	Project name Software Requirements Traceability Matrix	

## Project name Software Requirements Traceability Matrix

Version	Description	Author	Date

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	2 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Requirements Traceability Matrix		

## REVISION SHEET

Revision number	Revision date	Change description

**MITRON**

		Date	26.05.2014
		Pages	3 / 6
Author		Review	
Revision		Approval	
Document description	Project name Software Requirements Traceability Matrix		

## CONTENTS

<b>1</b>	<b>Introduction</b> .....	<b>2</b>
1.1	Purpose .....	2
1.2	Scope .....	2
1.3	Document organization.....	2
<b>2</b>	<b>References</b> .....	<b>2</b>
<b>3</b>	<b>Traceability</b> .....	<b>2</b>

---

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	4 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Requirements Traceability Matrix		

## LIST OF FIGURES

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	5 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Requirements Traceability Matrix		

## LIST OF TABLES

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	6 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Requirements Traceability Matrix		

## 1 Introduction

### 1.1 Purpose

### 1.2 Scope

### 1.3 Document organization

## 2 References

## 3 Traceability

# LIITE E: STPR POHJA

**MITRON**

		Date	26.05.2014
		Pages	1 / 6
Author		Review	
Revision		Approval	
Document description	Project name Software Test Procedure		

Project name  
Software Test Procedure

Version	Description	Author	Date

**MITRON**

		Date	26.05.2014
		Pages	2 / 6
Author		Review	
Revision		Approval	
Document description	Project name Software Test Procedure		

## REVISION SHEET

Revision number	Revision date	Change description

**MITRON**

	<b>Date</b>	26.05.2014
	<b>Pages</b>	3 / 6
<b>Author</b>		<b>Review</b>
<b>Revision</b>		<b>Approval</b>
<b>Document description</b>	Project name Software Test Procedure	

## CONTENTS

<b>1</b>	<b>Test procedure specification identifier .....</b>	<b>3</b>
<b>2</b>	<b>Purpose .....</b>	<b>3</b>
<b>3</b>	<b>Special requirements .....</b>	<b>3</b>
<b>4</b>	<b>Procedure steps .....</b>	<b>3</b>
4.1	Log .....	3
4.2	Set up .....	3
4.3	Start .....	3
4.4	Proceed .....	3
4.5	Measure	
	3	
4.6	Shutdown .....	3
4.7	Restart .....	3
4.8	Stop .....	3
4.9	Wrap up .....	3
4.10	Contingencies .....	3

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	4 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Test Procedure		

## LIST OF FIGURES

---

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	5 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Test Procedure		

## LIST OF TABLES

**MITRON**

		<b>Date</b>	26.05.2014
		<b>Pages</b>	6 / 6
<b>Author</b>		<b>Review</b>	
<b>Revision</b>		<b>Approval</b>	
<b>Document description</b>	Project name Software Test Procedure		

## 1 Test procedure specification identifier

## 2 Purpose

## 3 Special requirements

## 4 Procedure steps

4.1 Log

4.2 Set up

4.3 Start

4.4 Proceed

4.5 Measure

4.6 Shutdown

4.7 Restart

4.8 Stop

4.9 Wrap up

4.10 Contingencies