Tampereen ammattikorkeakoulu

Rinnakkaisten ohjelmistoversioiden vaatimustenhallinta Polarionissa

Roosa Laakkonen

OPINNÄYTETYÖ
Marraskuu 2024

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

LAAKKONEN, ROOSA:
Rinnakkaisten ohjelmistoversioiden vaatimustenhallinta Polarionissa

Opinnäytetyö 30 sivua
Marraskuu 2024

Opinnäytetyössä perehdyttiin vaatimustenhallinnan teoriaan ja tutkittiin rinnakkaisten ohjelmistoversioiden hallintaa Polarion ALM -työkalun avulla. Työ toteutettiin toimeksiantona yritykselle, joka tarvitsi työntekijöilleen oppaan Polarionin käyttöön. Oppaan sisältö rajattiin rinnakkaisten ohjelmistoversioiden hallintaan ja siihen liittyviin peruskäsitteisiin. Tavoitteena oli varmistaa, että työntekijät osaavat hyödyntää Polarionin keskeisiä ominaisuuksia vaatimustenhallinnan eri vaiheissa.

Teoreettisessa osuudessa käsitellään vaatimustenhallinnan keskeisiä käsitteitä, rinnakkaisten ohjelmistoversioiden hallinnan haasteita sekä DO-178C-standardin ohjelmistokehitykselle asettamia vaatimuksia. Käytännön osuudessa tutkitaan ja analysoidaan Polarionin tarjoamia mahdollisuuksia rinnakkain kehitettävien ohjelmistoversioiden vaatimustenhallintaan sekä laaditaan opas, joka auttaa työntekijöitä käyttämään Polarionin eri ominaisuuksia.

Työn tuloksena syntyi selkeä ja käyttäjäystävällinen opas, joka vastasi yrityksen tarpeisiin ja helpotti uusien käyttäjien siirtymistä Polarionin käyttöön. Oppaan arvioidaan parantavan organisaation vaatimustenhallintaa ja sisäisiä toimintamalleja. Johtopäätöksenä todetaan, että Polarionin monista laadukkaista ominaisuuksia huolimatta rinnakkaisten ohjelmistoversioiden vaatimustenhallinta ei ole täysin yksinkertaista. Inhimillisten riskien määrä on suuri, mutta automaation lisääntyminen voi tulevaisuudessa vähentää näitä riskejä.

Asiasanat: vaatimustenhallinta, rinnakkaiset ohjelmistoversiot, Polarion, DO-178C, ohjelmistokehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology Engineer
Software Engineering

LAAKKONEN, ROOSA:
Managing the Requirements of Parallel Software Versions in Polarion

Bachelor's thesis 30 pages
November 2024

This thesis focused on requirements management and examined the management of parallel software versions using the Polarion ALM tool. The work was commissioned by a company that needed a manual to help its employees use Polarion. The goal was to make sure the employees can effectively use Polarion's main features in the different stages of requirements management.

The theoretical part explained the key ideas of requirements management, the challenges of managing parallel versions, and the DO-178C standard for software development. The practical part explored how Polarion can help with these challenges and provided a simple manual for the employees.

The manual is expected to improve the company’s requirements management and internal practices. Although Polarion offers many good features, managing requirements for parallel versions remains complex. Human errors pose risks, though increasing automation may reduce these risks over time.

Key words: requirements management, parallel software releases, Polarion, DO-178C, software development

SISÄLLYS

LYHENTEET JA TERMIT

DAL	Development Assurance Levels
DO-178C	Software Considerations in Airborne Systems and Equipment Certification
FRD	Functional Requirement Document
Git	Versionhallintajärjestelmä
HLD	High Level Design
JIRA	Tehtävienhallintaohjelmisto
LLD	Low Level Design
Polarion ALM	Polarion® Application Lifecycle Management
SDD	Software Design Document
SRS	Software Requirements Specification

1 JOHDANTO

Vaatimustenhallinta on keskeinen osa ohjelmistokehitystä. Se korostuu erityisesti, kun kehitetään kriittisiä tai laajoja järjestelmiä. Vaatimustenhallinnan avulla voidaan varmistaa, että ohjelmiston ominaisuudet ja toiminnot vastaavat toiminnallisia tarpeita sekä teknisiä vaatimuksia. Puutteet tällä osa-alueella voivat johtaa ohjelmiston virheisiin ja turvallisuusriskeihin.

Ohjelmistosta voi myös olla samanaikaisesti kehityksessä useita eri versioita, mikä luo haasteita rinnakkaisten vaatimusten hallintaan. Rinnakkaisten versioiden hallinta vaatii tarkkaa suunnittelua ja työkaluja, jotka tukevat mm. vaatimusten jäljitettävyyttä ja versionhallintaa.

Tämän opinnäytetyön tavoitteena on tutkia rinnakkaisten ohjelmistoversioiden vaatimustenhallintaa teoreettisesta ja käytännönläheisestä näkökulmasta. Työ on toteutettu toimeksiantona yritykselle, joka tarvitsi oppaan työntekijöilleen Polarion ALM -työkalun käyttöön. Oppaan sisältö rajattiin yrityksen toiveiden mukaan koskemaan Polarionin peruskäsitteitä sekä rinnakkaisten ohjelmistoversioiden hallintaa. Tavoitteena on varmistaa, että työntekijät osaavat hyödyntää Polarionin keskeisiä ominaisuuksia vaatimustenhallinnan eri vaiheissa.

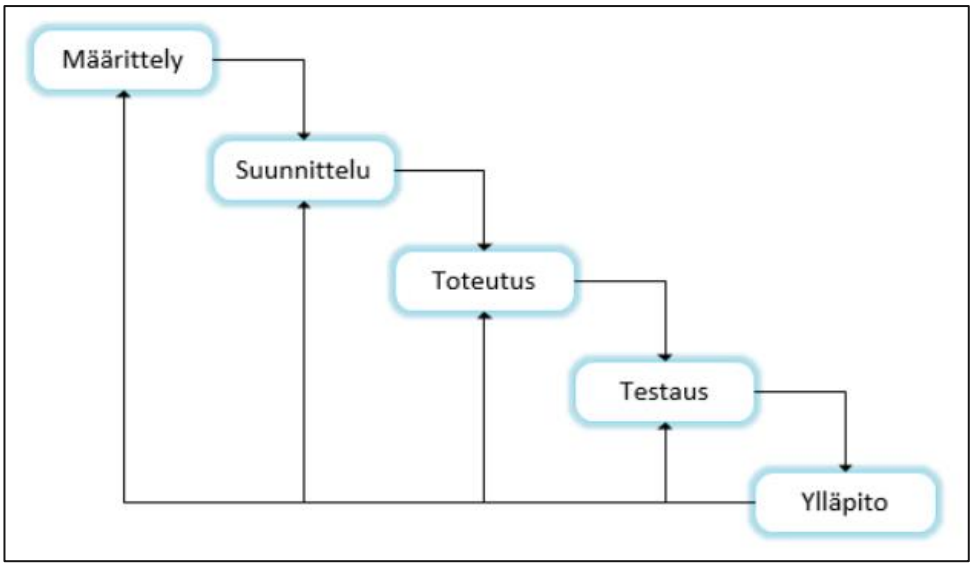
Työn teoreettisessa osuudessa käydään läpi vaatimustenhallinnan peruskäsitteitä, rinnakkaisten ohjelmistoversioiden hallinnan haasteita ja DO-178C-standardin ohjelmistokehitykselle asettamia vaatimuksia. Opinnäytetyön käytännönosuutena tutkitaan ja analysoidaan Polarionin tarjoamia mahdollisuuksia rinnakkaisten ohjelmistoversioiden vaatimusten hallintaan sekä laaditaan Polarionin käyttäjille opas.

2 VAATIMUSTENHALLINTA

Ohjelmistokehityksessä vaatimuksella tarkoitetaan jonkin dokumentoidun tarpeen, odotuksen tai pakollisuuden kuvausta, joka ohjelmiston tai järjestelmän tulee täyttää. Vaatimukset ovat ohjelmistosuunnittelun ja –kehityksen perusta, sillä ne määrittävät, mitä ohjelmiston tulee tehdä ja miten sen tulee käyttäytyä. (Sommerville 2011, 83.) Ohjelmistokehityksen vaihetta, jossa määritellään ohjelmiston täytettävät vaatimukset, kutsutaan vaatimusmäärittelyksi (Requirement Engineering 2024).

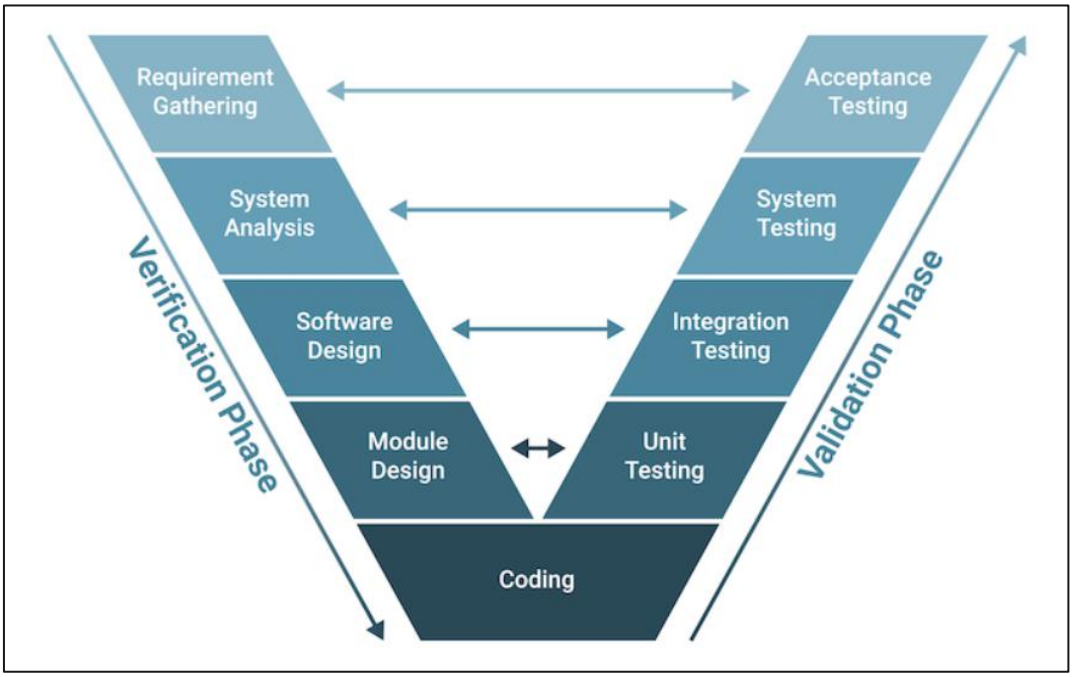
Vaatimustenhallinta on prosessi, jossa kerätään, määritellään, analysoidaan ja dokumentoidaan projektin tai järjestelmän vaatimuksia (Requirement Engineering 2024). Se on mukana ohjelmistokehityksen eri elinkaaren vaiheissa, eli suunnittelussa, toteutuksessa, testaamisessa, integroinnissa ja ylläpidossa. Yleensä kaikki projektit edellyttävät jonkinlaisia muutoksia kehityksen edetessä. Vaatimusten huolellinen määrittely ja dokumentointi auttaa välttämään väärinymmärryksiä sekä varmistamaan, että kaikki sidosryhmät pysyvät kartalla työn etenemisestä läpi projektin. Lisäksi se helpottaa myös analysoimaan muutoksia, jotta projekti pysyy asiakkaan toiveiden mukaisena. (Sommerville 2011, 84.)

Vaatimustenhallinnan rooli ja toteutustapa voivat vaihdella kehitysmenetelmän mukaan. Vesiputousmallissa (KUVA 1.) vaatimustenhallinta korostuu erityisesti kehityksen alkuvaiheessa, kun vaatimukset pyritään määrittelemään mahdollisimman kattavasti ennen siirtymistä seuraaviin vaiheisiin. Mallin mukaisessa kehityksessä muutokset kehityksen myöhemmissä vaiheissa voivat olla kalliita tai monimutkaisia, koska paluu edellisiin vaiheisiin ei ole suoraviivaista. (Sommerville 2011, 30.)



KUVA 1. Ohjelmistokehityksen vesiputousmalli (Sommerville 2011, 30)

Toinen yleisesti käytetty ohjelmistokehitysmalli on V-malli (KUVA 2.). Se keskittyy vesiputousmallia perusteellisemmin verifiointiin ja validointiin. V-mallissa vaatimustenhallinta kytkeytyy suoraan testaukseen ja dokumentointi on osa jokaista vaihetta (Sommerville 2011, 42.) Jatkuva suunnittelu ja dokumentointi on aikaa vievää, ja voi hidastaa kehitysprosessia. V-malli onkin yleensä parempi vaihtoehto, kun projektin laadunvarmistus ja riskienhallinta ovat erityisen tärkeässä roolissa.



KUVA 2. Ohjelmistokehityksen V-malli (What Is the V-Model in Software Development? 2023)

2.1 Vaatimustyypit

Ohjelmistosuunnittelun vaatimukset voidaan jakaa yleisesti toiminnallisiin ja ei-toiminnallisiin vaatimuksiin (TAULUKKO 1.). Ne määrittelevät ohjelmiston toimintatavat ja laadulliset ominaisuudet. Vaatimukset voivat ottaa kantaa esimerkiksi suorituskykyyn, tietoturvaan tai käytettävyyteen. Toiminnalliset vaatimukset kuvaavat järjestelmän keskeisiä toimintoja käyttäjän näkökulmasta, ja ei-toiminnalliset vaatimukset keskittyvät järjestelmän laatuun. Lisäksi käytetään testitapauksia, jotka varmistavat, että ohjelmisto täyttää asetetut vaatimukset käytännössä. (Sommerville 2011, 85.)

TAULUKKO 1. Toiminnallisten ja ei-toiminnallisten vaatimusten eroja (Functional vs Non-Functional Requirements 2024)

Toiminnalliset vaatimukset	**Ei-toiminnalliset vaatimukset**
Määrittelee järjestelmän tai sen osan.	Määrittelee ohjelmistojärjestelmän laatuominaisuuden.
Määrittelee ”Mitä ohjelmistojärjestelmän pitäisi tehdä?”.	Asettaa rajoituksia: ”Miten ohjelmistojärjestelmän pitäisi täyttää toiminnalliset vaatimukset?”
Käyttäjä määrittelee.	Tekniset henkilöt, kuten arkkitehti, tekniset johtajat ja ohjelmistokehittäjät määrittelevät.
Pakollinen.	Ei pakollinen.
Auttaa todentamaan ohjelmiston toimivuuden.	Auttaa todentamaan ohjelmiston suorituskyvyn.
Yleensä helppo määritellä.	Yleensä vaikeampi määritellä.

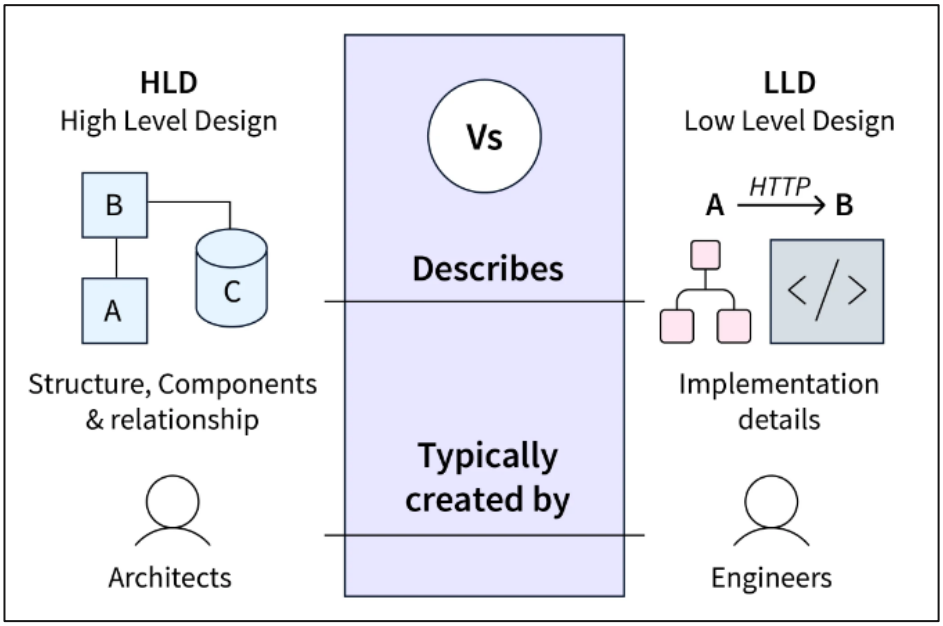
2.1.1 Korkean ja matalan tason vaatimukset

Vaatimukset voidaan jakaa myös korkean ja matalan tason vaatimuksiin. Korkean tason vaatimukset viittaavat yleensä laajempiin, strategisiin tarpeisiin. Ne kuvaavat, mitä järjestelmän tai ohjelmiston on saavutettava kokonaisuudessaan. Korkean tason vaatimukset tulevat usein suoraan asiakkaalta tai projektin sidosryhmiltä. Korkean tason vaatimuksista vastaa yleensä projektin johto tai pääarkkitehdit, jotka määrittävät projektin suunnan. (Ohjelmistojen vaatimusmäärittely

2011, 160.) Esimerkiksi vaatimus ”Käyttäjän on voitava lähettää viesti toiselle käyttäjälle.” olisi korkean tason vaatimus.

Matalan tason vaatimukset ovat tarkempia ja yksityiskohtaisempia, ja ne liittyvät siihen, miten korkean tason vaatimukset toteutetaan käytännössä. Vaatimuksista vastaavat yleensä ohjelmistokehittäjät tai suunnittelutiimit, joiden tehtävä on huolehtia konkreettisista toteutuksista. Matalan tason vaatimukset täydentävät korkeaa tasoa ja määrittävät tarkemmin, millaisia toiminnallisuuksia ja elementtejä tarvitaan korkean tason vaatimusten saavuttamiseksi. (Ohjelmistojen vaatimusmäärittely 2011, 160.) Esimerkiksi vaatimus ”Sovelluksessa on oltava *Lähetä*-painike.” olisi matalan tason vaatimus.

KUVA 3 havainnollistaa korkean ja matalan tason suunnittelun eroja ja vastuualueita. Korkean tason suunnittelu (High Level Design, HLD) kattaa järjestelmän rakenteen ja pääkomponenttien väliset suhteet. Se on usein arkkitehtien vastuulla. Matalan tason suunnittelu (Low Level Design, LLD) keskittyy tarkempiin toteutusyksityiskohtiin ja siitä vastaavat pääosin insinöörit.



KUVA 3. Korkean ja matalan tason vaatimusten erot havainnollistettuna (Difference between High-Level and Low-Level Design 2023)

2.2 Vaatimusten dokumentointi

Dokumentaatio tarkoittaa kirjallisia tai digitaalisia materiaaleja, jotka tallentavat ja kuvaavat ohjelmiston kehitykseen, rakenteeseen, toimintaan ja ylläpitoon liittyviä tietoja. Dokumentointi toimii todisteena vaatimusten täyttymisestä ja kehitysprosessin asianmukaisuudesta. Vaatimusdokumentteja on useita erilaisia. Ne kuvaavat ohjelmiston tai järjestelmän toiminnallisia ja teknisiä tarpeita. Kolme yleisesti käytettyä dokumenttia ovat FRD (Functional Requirements Document), SRS (Software Requirements Specification) ja SDD (Software Design Document) (Sommerville 2011, 90.)

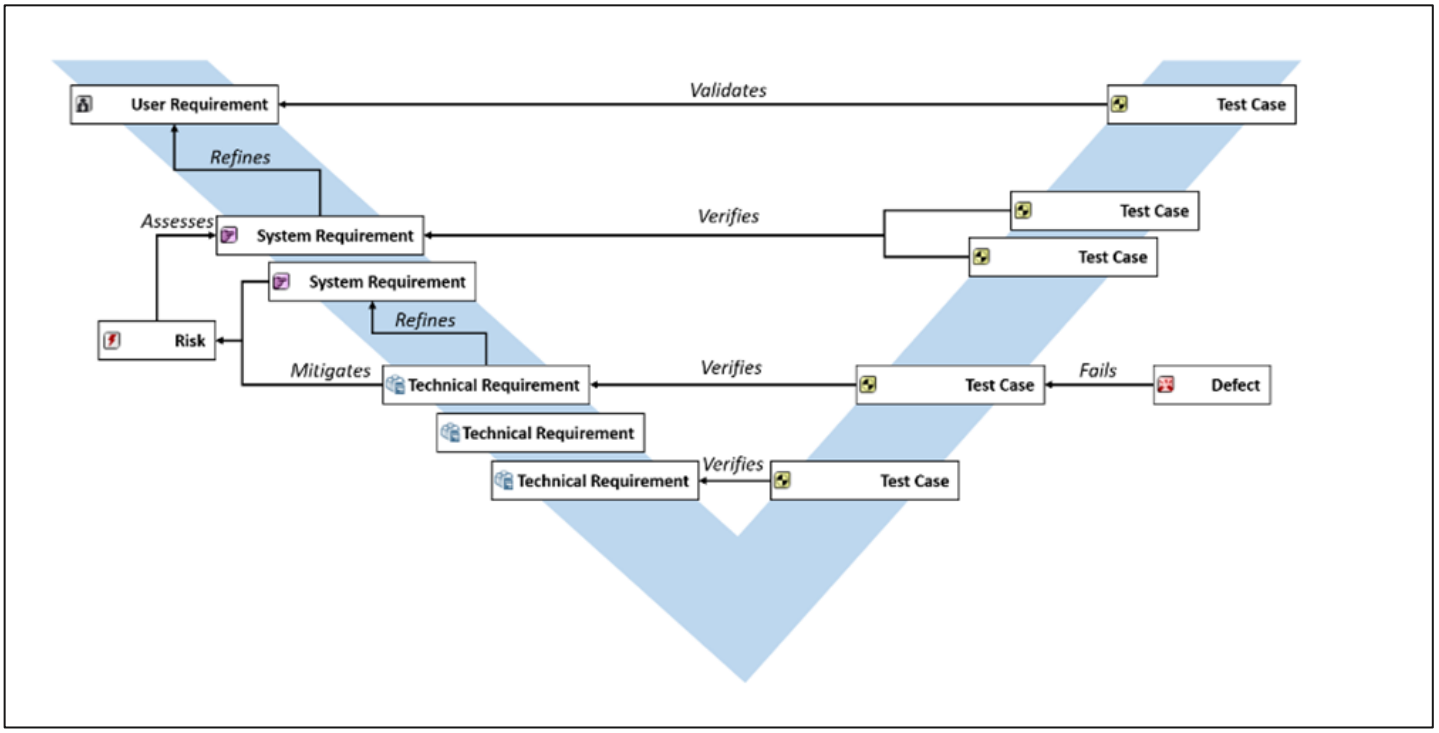
FRD keskittyy toiminnallisiin vaatimuksiin. Se kertoo, mitä ominaisuuksia ja toimintoja käyttäjä voi odottaa ohjelmistolta. FRD auttaa kehittäjiä ymmärtämään, millaisia ratkaisuja heidän täytyy luoda käyttäjän tarpeiden ja odotusten täyttämiseksi. (Sommerville 2011, 325.) SRS on yleensä FRD:tä laajempi dokumentti, joka sisältää kaikki ohjelmiston vaatimukset, sekä toiminnalliset että ei-toiminnalliset. Se tarjoaa tarkemman ja kattavamman kuvauksen siitä, mitä teknisiä ehtoja ohjelman on täytettävä. (Sommerville 2011, 91.) SDD puolestaan keskittyy ohjelmiston tekniseen suunnitteluun. Siinä määritellään tarkasti, kuinka ohjelmiston eri osat toteutetaan vastaamaan SRS:n ja FRD:n vaatimuksia. (Sommerville 2011, 93.)

Yksinkertaistetusti voi ajatella, että FRD on näistä kolmesta käyttäjälähtöisin. Se kuvaa, mitä järjestelmältä odotetaan, kun SRS kertoo syvällisemmin, miten nämä vaatimukset saavutetaan teknisesti. FRD ja SRS voivat molemmat sisältää sekä korkean että matalan tason vaatimuksia, mutta FRD painottuu käyttäjän näkökulmaan ja SRS tekniseen toteutukseen. SDD puolestaan kokoaa nämä vaatimukset yhteen konkreettiseksi tekniseksi suunnitelmaksi. (Sommerville 2011, 90, 325.)

2.3 Jäljitettävyys

Jäljitettävyydellä tarkoitetaan ohjelmistokehityksessä kykyä seurata ja havaita eri vaatimusten ja muutosten välisiä yhteyksiä. Se helpottaa ymmärtämään, miten

vaatimukset on käännetty teknisiksi ratkaisuiksi, ja auttaa seuraamaan, kuinka ohjelmiston ominaisuudet ja muutokset vaikuttavat sen muihin osiin, esimerkiksi koodiin, testeihin sekä dokumentaatioon (KUVA 4.). (Sommerville 2011, 113.)



KUVA 4. Esimerkki vaatimusten ja testien jäljitettävyydestä (Administrator and User Help… 2023)

Jäljitettävyyden avulla pyritään varmistamaan, että kaikki vaatimukset on huomioitu ja testattu. Sen tarkoituksena on mahdollistaa paremman kokonaisuuden hallinta ja auttaa kehittäjiä arvioimaan, mihin kaikkialle yksittäinen muutos voi vaikuttaa. Sen avulla halutaan minimoida virheiden riskiä ja varmistaa, että ohjelmisto toimii odotetulla tavalla myös päivitysten jälkeen. (DO-178C Software Development for Safety Critical Systems 2023.) On kuitenkin huomionarvoista, että jäljitettävyys seuraa nimenomaan yhteyksiä, eikä osaa ottaa kantaa dokumenttien sisältöön. Jäljitettävyyden osalta kaikki voi näyttää olevan kunnossa, vaikka esimerkiksi vaatimusten sisältö olisikin heikkoa.

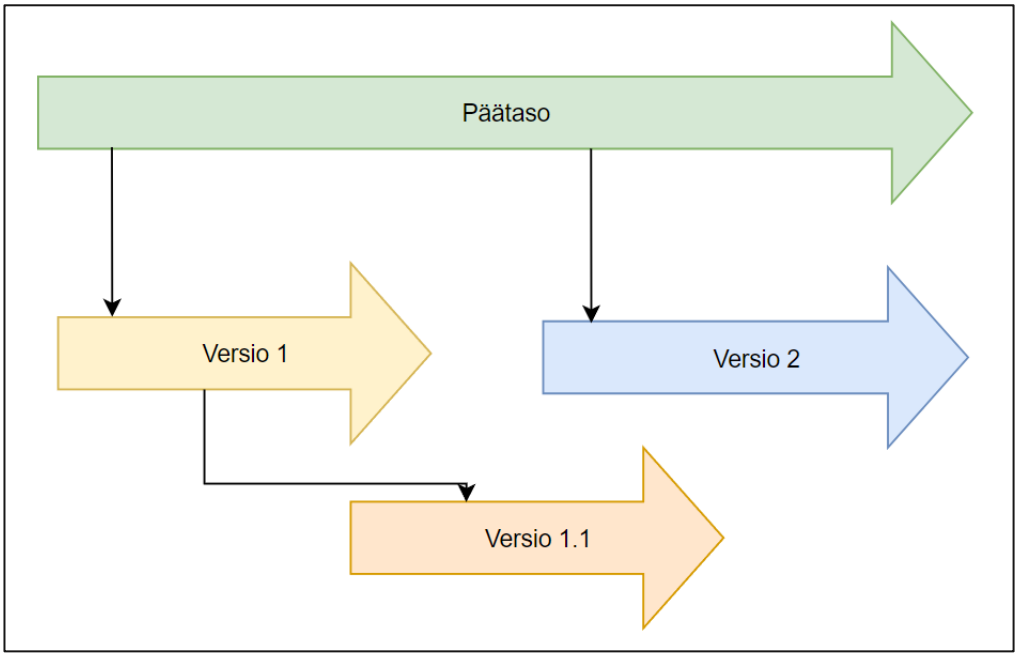
Täydellinen jäljitettävyys mahdollistaa kaikkien vaatimusten, suunnitelmien, toteutusten, testien ja muutoksien täyden jäljittämisen toisiinsa. Se tarkoittaa, että jokaiselle vaatimukselle löytyy siihen liittyvä suunnitteludokumentaatio, toteutettu koodirivi, testitapaus ja muutokset, joita on tehty kehityksen aikana. Täydellinen jäljitettävyys on tärkeää turvallisuuskriittisissä ja suurissa projekteissa, joissa on

tiukat laatu- ja turvallisuusvaatimukset. (DO-178C Software Development for Safety Critical Systems 2023.) Pienemmissä, ketterämmissä projekteissa tämän tasoinen jäljitettävyys ei ole välttämätöntä. Vähäinenkin jäljitettävyys on kuitenkin suositeltavaa, sillä se parantaa projektinhallintaa.

Jäljitettävyyden seuraamiselle on kehitetty erilaisia työkaluja. Työkaluja on useita erilaisia, ja yksi suosituimmista on jäljitettävyysmatriisi (traceability matrix). Se esittää selkeästi, kuinka eri vaatimukset liittyvät toisiinsa sekä muihin projektin osiin, esim. testitapauksiin ja toteutuksiin. Jäljitettävyysmatriisi visualisoi yhteyksiä ja pyrkii varmistamaan, että kaikki vaatimukset on huomioitu ja testattu asianmukaisesti. (Sommerville 2011, 115.)

3 RINNAKKAISET OHJELMISTOVERSIOT

Samaa ohjelmistoa voidaan kehittää useampana versiona yhtä aikaa. Rinnakkaisten ohjelmistoversioiden käyttö on yleistä varsinkin isoissa projekteissa, joissa ohjelmisto tarvitsee uusia ominaisuuksia ja korjauksia vanhempiin versioihin. Ohjelmistoversioiden rinnakkaisuus siis tarkoittaa, että useat tiimit tai kehittäjät työskentelevät samaan aikaan, mutta eri ohjelmistoversioiden parissa. Yksi tiimi voi esimerkiksi kehittää uutta versiota ja uusia ominaisuuksia, ja toinen tiimi tekee virheenkorjauksia vanhempaan versioon, joka on jo käytössä (KUVA 5.). Rinnakkaisuus tekee ohjelmiston kehittämisestä tehokkaampaa, mutta vaatii tarkkaa hallintaa, jotta muutokset ja korjaukset pysyvät johdonmukaisina eri versioiden välillä. (Systems Engineering 2023.)



KUVA 5. Yksinkertaistettu esimerkki ohjelmistoversioiden kehityksestä rinnakkain

3.1 Hyödyt

Usean ohjelmistoversion kehittäminen rinnakkain tuo mukanaan hyötyjä. Yksi merkittävimmistä eduista on mahdollisuus hyödyntää resursseja tehokkaammin.

Kun useat tiimit voivat työskennellä samanaikaisesti eri versioiden parissa, kehitysprosessi nopeutuu, ja uusien ominaisuuksien nopeampi käyttöönotto on mahdollista. (Systems Engineering 2023.)

Rinnakkaisuus tarjoaa myös mahdollisuuden kokeilla uusia ideoita ilman, että se vaikuttaa tuotannossa olevaan versioon. Tiimit voivat esimerkiksi kehittää kokeellisia ominaisuuksia, ja testata niitä ennen lopullista päätöstä niiden integroimisesta pääversioon. Tällainen mahdollisuus innovatiiviseen työskentelyyn on arvokas ja voi johtaa merkittäviin edistysaskeliin.

3.2 Haitat

Rinnakkaisia ohjelmistoversioita kehitettäessä vaatimukset tai testit voivat muuttua vain yhdessä versiossa ja pysyä ennallaan toisessa. Tällainen ominaisuuksien haarautuminen voi johtaa epäjohdonmukaisiin tai ristiriitaisiin vaatimuksiin, ja vaikeuttaa vaatimustenhallintaa. Ohjelmiston eri osat ovat kuin palapelin palasia: kun yhtä osaa muutetaan, se voi muuttaa muiden osien toimintaa. Vaatimuksia hallitessa, yhden vaatimuksen muutos voi heijastua siis koko järjestelmään. Ilman huolellisia raportointityökaluja voi olla haastavaa seurata, mitkä vaatimukset ja koodimuutokset liittyvät mihinkin versioon. (Systems Engineering 2023.)

Lisäksi dokumentaatio voi nopeasti hajautua ja ajautua ristiriitoihin, kun useita versioita kehitetään samanaikaisesti. Tiettyihin vaatimuksiin ja toiminnallisuuksiin liittyvät muutokset voivat jäädä päivittämättä toiseen versioon tai dokumentointi voi muuttua yhdestä versiosta toiseen ilman asianmukaista seurantaa. On myös huomioitavaa, että rinnakkaiset versiot edellyttävät, että jokainen ohjelmistoversio verifioidaan ja validoidaan erikseen. (Systems Engineering 2023.)

4 DO-178C-STANDARDI

Ilmailuun liittyvissä projekteissa on erittäin tarkat vaatimukset, jotka ohjelman tai ominaisuuden pitää täyttää. DO-178C on siviili-ilmailun ohjelmistokehitykseen ja sertifiointiin kehitetty, vuonna 2011 julkaistu, standardi. Se määrittelee vaatimukset ohjelmiston kehitysprosessille, joita noudattamalla voidaan varmistaa, että ohjelmisto täyttää turvallisuusvaatimukset ja on riittävän luotettava käytettäväksi ilmailualalla. (DO-178C Software Development for Safety Critical Systems 2023.)

Standardi koskee kaikkia ohjelmistotyyppejä, jotka vaikuttavat suoraan tai välillisesti lentokoneen turvallisuuteen ja sitä sovelletaan ohjelmistojen kehityksessä, ylläpidossa sekä sertifioinnissa. Standardin tavoitteiden noudattaminen on välttämätöntä ilmailun sääntelyvaatimusten täyttämiseksi ja sääntelyviranomaisten hyväksynnän saamiseksi. (DO-178C Software Development for Safety Critical Systems 2023.)

4.1 Ohjelmiston kriittisyystaso

Standardi jakaa vaatimukset viidelle kriittisyystasolle. (Development Assurance Levels, DAL) A:sta E:hen, joista Taso A on kriittisin ja Taso E vähiten kriittinen (TAULUKKO 2). Taso määrittää, kuinka perusteellisesti ohjelmiston kehitysprosessia ja tuloksia on tarkasteltava ja dokumentoitava. (DO-178C Software Development for Safety Critical Systems 2023.)

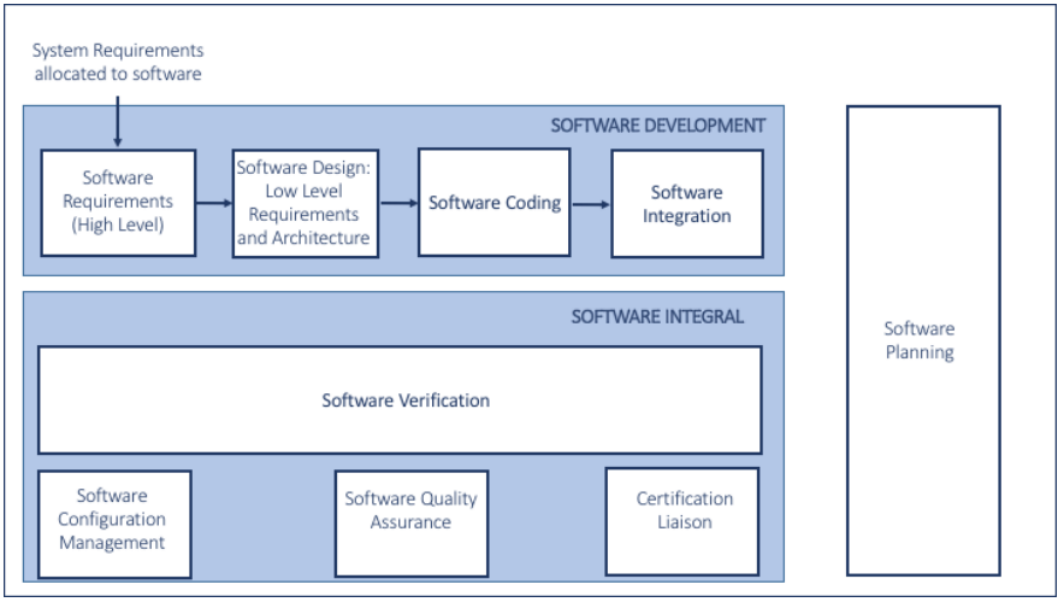
TAULUKKO 2. DAL-kriittisyystasot

Taso	**Seuraus**
A (Katastrofaalinen)	Vika johtaa katastrofaaliseen tapahtumaan, kuten lentokoneen menetykseen tai vakaviin henkilövahinkoihin (esim. kuolemantapaukset)
B (Vaarallinen)	Vika johtaa vakavaan vaaratilanteeseen, vakaviin vammoihin tai kuolemaan.
C (Mahdollisesti vaarallinen)	Vika aiheuttaa merkittävän vikatilanteen tai vammoja.
D (Vähäinen)	Vika aiheuttaa vähäisen vikatilanteen tai epämukavuutta.
E (Ei vaikutusta turvallisuuteen)	Vika ei vaikuta lentokoneen tai matkustajien turvallisuuteen.

Kriittisyystasot eivät ainoastaan vaikuta vaatimusten tiukkuuteen, vaan myös ohjaavat kehitystiimejä resurssoinnissa ja aikarajoissa. Esimerkiksi korkeamman kriittisyystason ohjelmistot vaativat enemmän aikaa ja resursseja kehityksen eri vaiheissa, mikä vaikuttaa projektin aikatauluun ja budjettiin. (Systems Engineering 2023.)

4.2 Standardin vaikutus elinkaareen

Standardi ohjaa koko ohjelmiston kehitysprosessia, jotta ohjelmisto täyttää kaikki turvallisuusvaatimukset (KUVA 6.). Standardi vaatii, että kaikkien ohjelmiston vaatimusten tulee olla selkeitä, yksiselitteisiä ja testattavia. Lisäksi vaatimusten tulee kattaa kaikki ohjelmiston toiminnot ja turvallisuusnäkökohdat. (DO-178C Software Development for Safety Critical Systems 2023.)



KUVA 6. DO-178C-standardin määrittelemä ohjelmiston kehitysprosessi (DO178C Software Life Cycle Processes 2017)

Kaikki kehitysprosessin vaiheet dokumentoidaan; suunnitelmat, vaatimukset, muutokset, virheet jne. Jatkuva dokumentointi lisää hallittavien artefaktien, eli tuotosten määrää, ja korostaa siten myös jäljitettävyyden tärkeyttä. (Systems Engineering 2023.) DO-178C vaatii täyden jäljitettävyyden. Mahdollisuus jäljittää jokainen ohjelmiston vaatimus sen suunnitteluun, toteutukseen ja testaukseen on välttämätöntä, jotta voidaan varmistaa, että kaikki vaatimukset on täytetty ja mahdolliset puutteet tai virheet voidaan nopeasti paikantaa ja korjata. Kaikista muutoksista tulee jäädä jälki rekisteriin ja vanhoihin versioihin tulee olla mahdollista palata. (DO-178C Software Development for Safety Critical Systems 2023.)

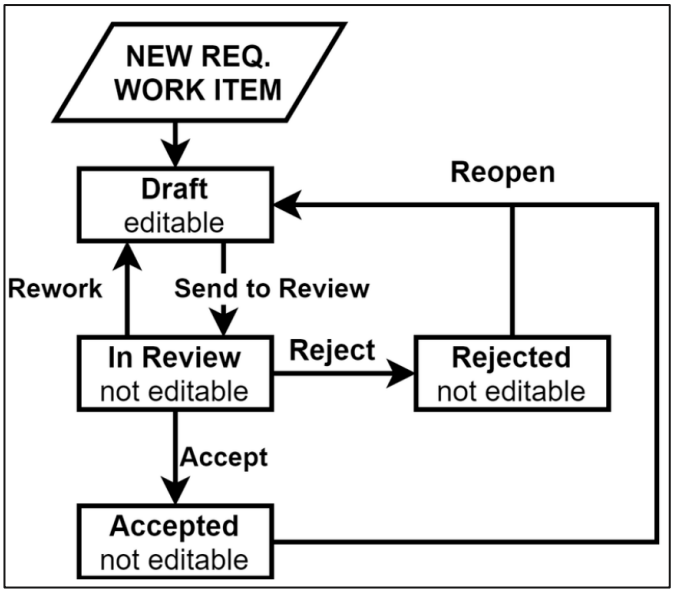
Ohjelmiston kriittisyyden taso (DAL) määrittää testauksen laajuuden ja syvyyden. Mitä korkeampi kriittisyystaso, sitä enemmän vaaditaan todisteita siitä, että ohjelmisto toimii oikein myös äärriolosuhteissa ja virhetilanteissa. (DO-178C Software Development for Safety Critical Systems 2023.) Ylimääräinen testaaminen ja dokumentointi todennäköisesti pidentää kehitysaikaa ja lisää kustannuksia, mutta on tarpeellista, jotta saavutetaan tarvittava luotettavuus ja turvallisuus.

5 POLARION ALM

Polarion ALM (Application Lifecycle Management) on Siemensin kehittämä pilvipohjainen työkalu. Se kerää kaikki ohjelmistokehityksen vaiheet yhteen integroituun ympäristöön. Polarion on yhtenäinen ratkaisu testien- ja riskienhallintaan sekä laadunvarmistukseen. Lisäksi se tukee eritasoisia vaatimusstandardeja ja kykenee siten vastaamaan esimerkiksi ilmailun tarpeisiin. (Polarion ALM 2024.) Tyypillisiä ohjelman käyttäjiä ovat ohjelmistokehittäjät, laadunvarmistusinsinöörit, projektipäälliköt sekä vaatimustenhallinnan asiantuntijat.

Polarionin käyttäjät voivat luoda, muokata ja seurata vaatimuksia, testejä ja dokumentteja reaaliajassa. Sovelluksen kaikki dokumentit ja tiedot ovat versioituja, jolloin eri tiimit tai tiimin jäsenet voivat työskennellä yhdessä ilman tietojen häviämistä. (Polarion® ALM™ 2024.)

Polarion käyttää kehitysprosessin elementeistä nimeä ”Work Item” (suom. työkohde). Työkohteet voivat olla erityyppisiä, ja niillä voi olla erilaiset attribuutit (esim. tila tai vastuuhenkilö). Tällaisia työkohteita ovat esimerkiksi vaatimukset ja testit. Uuden työkohteen luominen on tarkkaan määritelty prosessi, ja se etenee vaiheittain luonnoksesta hyväksyntään (KUVA 7.).



KUVA 7. Uuden työkohteen hyväksyntäprosessi (A Requirements Management Template in Polarion… 2021)

Kaikista työkohteista ja niiden muutoksista jää historiaan jälki. Lisäksi jokaisen muutoksen jälkeen tehdyn tallennuksen myötä Polarion luo kyseiselle työkohteelle uuden revision. Revisioita voidaan myöhemmin selata, ja näin ollen myös tarkastella miltä mikäkin dokumentti tai vaatimus on tiettynä ajanhetkenä näyttänyt. Ohjelma mahdollistaa näin täyden muutoshistorian säilymisen. Lisäksi Polarion tukee täyttä jäljitettävyyttä, ja projektin jokainen vaihe voidaan yhdistää alkuperäisiin vaatimuksiin.

Polarion sisältää myös kattavat raportointi- ja analytiikkatyökalut, joiden avulla käyttäjät saavat dataa projektin edistymisestä ja laadusta. Työkalu mahdollistaa lisäksi laajan integroinnin muiden kehitystyökalujen kanssa, esimerkiksi JIRA:n ja Git:in kanssa. Juuri näistä syistä Polarion oli otettu käyttöön myös tämän työn toimeksiantajan organisaatiossa. Sen tarjoamat ominaisuudet vastasivat heidän vaatimustenhallinnan tarpeisiinsa.

6 OPPAAN LAATIMINEN

Opinnäytetyön osana laadittiin opas Polarion ALM:n käyttäjille. Sen tarkoituksena oli tarjota selkeät ja käytännönläheiset ohjeet rinnakkaisten ohjelmistoversioiden hallintaan. Oppaan laatiminen perustui raportin teoreettiseen osaan sekä käytännön kokemuksiin Polarionin käytöstä.

6.1 Oppaan rakenteen ja sisällön suunnittelu

Oppaan laatiminen aloitettiin suunnitellulla. Suunnittelussa keskityttiin määrittelemään keskeiset käsiteltävät aihealueet. Polarion tarjoaa paljon erilaisia ominaisuuksia, eikä kaikkia ollut mitenkään mahdollista käydä läpi oppaassa, joten oppaan alue rajattiin nimenomaan rinnakkaisten ohjelmistoversioiden vaatimustenhallintaa koskeviin käsitteisiin. Tämän takia oppaassa ei oteta kantaa ollenkaan esimerkiksi testitapauksiin tai testien automatisointiin.

Opas kattaa vaatimustenhallintaa koskevat peruskäsitteet, ohjeistuksen päätason (Mainline) ja perustason (Baseline) hallinnointiin sekä haarauttamiseen (branching), dokumenttien vertailun ja yhdistämisen (merge). Rakenne pyrittiin luomaan loogiseksi eli vastaamaan työnkulkua (KUVA 8.). Ensimmäisenä esitellään perusterminologia, ja sen jälkeen käydään läpi yksityiskohtaisemmin Mainlinen ja Baselinen toiminta.

Contents:

KUVA 8. Oppaan sisällysluettelo

Oppaan alkuperäisenä tavoitteena oli luoda niin selkeät ohjeet, että kuka tahansa pystyisi käsittelemään rinnakkaisia vaatimuksia. Kuitenkin Polarionin aivan perustoimintojen selittäminen olisi paisuttanut oppaan liian laajaksi. Lopulta oppaan sisältö suunniteltiin siten, että kohderyhmänä olivat käyttäjät, joilta löytyi jo jonkin verran kokemusta Polarionin käytöstä. Lisäksi oppaan kieleksi valittiin englanti, jotta se olisi laajemmin käyttäjien saavutettavissa. Oppaassa pyrittiin käyttämään

selkeää kieltä ja tarjoamaan paljon visuaalisia esimerkkejä. Oppaan käyttöä helpotettiin myös sisällyttämällä yksinkertaisia ja helposti seurattavia vaiheittaisia ohjeita.

6.2 Oppaan kirjoittaminen

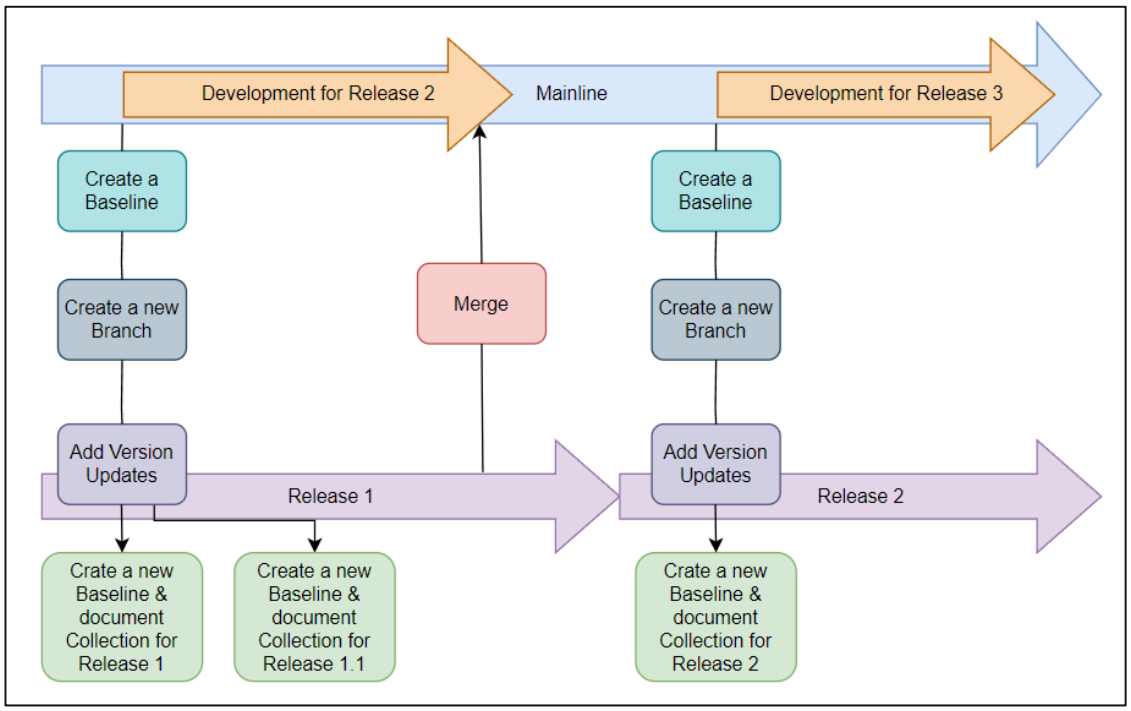
Kirjoittamisessa hyödynnettiin Polarionin omia käyttöoppaita, tutoriaalivideoita ja dokumentaatiota sekä Taipuva Consulting Oy:n luomia ohjeita. Lisäksi apuna käytettiin vaatimustenhallintaan liittyvää teoriaa sekä kirjoittajan kerryttämää kokemusta Polarionin käytöstä. Työ sisälsi myös käytännön kokeiluja Polarionissa, demovaatimusten ja -dokumenttien luomista sekä yhteydenpitoa asiantuntijan kanssa.

Oppaan kirjoittamisen yhteydessä sovelluksen ominaisuuksia testattiin ja niiden käyttöä harjoiteltiin Polarionin hiekkalaatikossa, eli harjoittelu- ja kokeiluympäristössä. Hiekkalaatikko mahdollistaa Polarionin toimintojen ja konfiguraatioiden testaamisen ilman, että varsinaiset projekti- tai tuotantoympäristön tiedot vaarantuvat. Myös oppaassa käytetyt demovaatimukset ja -dokumentit luotiin hiekkalaatikossa.

Oppaan luonnos esiteltiin käyttäjäryhmälle, ja siitä pyydettiin vapaamuotoista palautetta. Käyttäjäryhmään kuului eri tasoisia käyttäjiä, joiden näkemykset auttoivat kehittämään oppaan sisältöä monipuolisemmaksi. Palautteessa toivottiin selkeytystä Polarionin perusterminologiaan, toimintatapoja tukevia perusteluita sekä enemmän käytännönläheisiä esimerkkejä. Palautteen pohjalta oppaaseen tehtiin useita parannuksia: sisältöä tarkennettiin, tekstikappaleita muokattiin helpommin ymmärrettäviksi, ja oppaaseen lisättiin uusia kuvia sekä vaiheittaisia ohjeita. Muutoksilla pyrittiin varmistamaan, että oppaasta tulee käyttäjäystävällinen ja selkeä apuväline työntekijöille.

6.3 Rinnakkaisten ohjelmistoversioiden vaatimustenhallinta tiivistettynä

Oppaassa esitelty rinnakkaisten ohjelmistoversioiden hallintaprosessi koostuu useista vaiheista. Työ lähtee liikkeelle uuden ohjelmistoversion haarauttamisella pois Mainlinesta (pääversiosta), ja jatkuu haarautuneiden dokumenttien käsittelyllä (KUVA 9).



KUVA 9. Rinnakkaisten ohjelmistoversioiden vaatimustenhallintaprosessi

Tiivistettynä työnkulku on seuraavanlainen:

1. Mainlinesta luodaan Baseline

Ensimmäiseksi luodaan Baseline (perustaso/lähtötaso) Mainlineen tallennetuille dokumenteille. Käytännössä tämä tarkoittaa asiakirjojen, työkohteiden jne. jäädyttämistä täsmälleen siihen hetkeen (versioon), jolloin Baseline on tehty. Baselinen voi luoda koko projektille kerrallaan tai yksittäiselle asiakirjalle (suositus). Baseline toimii kiintopisteenä, johon dokumentteja voidaan verrata myöhemmin ja josta voidaan haarauttaa (branch) uusia versioita.

2. Haaran luominen

   Heti Baselinen luonnin jälkeen luodaan uusi haara dokumenteille. Tämä haara mahdollistaa rinnakkaisten versioiden kehittämisen siten, että muutokset pysyvät erillään päälinjasta (Mainline) ja kohdistuvat vain haluttuun ohjelmistoversioon.

3. Haarautettujen dokumenttien siirtäminen oikeaan kansioon

   Haarautetut dokumentit siirretään oikeaan kansioon, joka on nimetty version mukaan (esimerkiksi Versio 1.1). Tähän kansioon kuuluvat kaikki kyseiseen versioon liittyvä dokumentaatio ja testit. Hyvä kansiorakenne auttaa järjestämään dokumentit eri ohjelmistoversioiden mukaan ja vähentää sekaannusten riskiä.

4. Haarautettujen dokumenttien päivitys

   Seuraavaksi haarautettuja dokumentteja päivitetään niillä muutoksilla, jotka ovat tarpeen kyseiselle versiolle. Nämä muutokset jäävät haaran sisälle, eikä niitä päivitetä takaisin Mainlineen, mikä pitää pääversion muuttumattomana.

5. Haarautettujen dokumenttien hyväksyntä

   Kun kaikki muutokset on tehty ja haarautetut dokumentit ovat valmiita julkaisua varten, ne hyväksytään virallisesti.

6. Uuden Baselinen luominen hyväksytyille dokumenteille

   Jokaisesta hyväksytystä haarautetusta dokumentista luodaan uusi Baseline, joka kuvaa dokumentin tilaa julkaisun hetkellä.

7. Dokumenttien liittäminen kokoelmaan

   Lopuksi hyväksytyt ja Baselinen saaneet dokumentit liitetään kokoelmaan, joka on koottu kyseistä ohjelmistoversiota varten. Tämä varmistaa, että kaikki versioon liittyvät dokumentit ovat selkeästi hallittavissa ja jäljitettävissä.

8. Haluttujen uusien ominaisuuksien tai päivitysten sulauttaminen takaisin Mainlineen (valinnainen)

   Halutut uudet ominaisuudet, dokumentit ja vaatimukset voidaan mergetä takaisin Mainlineen. Tämä vaihe ei ole pakollinen, mutta se mahdollistaa kehitystyön aikana syntyneiden parannuksien käyttöönoton pääversiossa.

6.4 Haasteet

Polarion tarjoaa monipuolisesti erilaisia työkaluja vaatimustenhallintaan, mutta juuri tämän laajuuden takia työn tekeminen osoittautui ennakoitua työläämmäksi. Projektin edetessä tuli tarpeelliseksi perehtyä useisiin Polarionin eri toimintoihin, mikä vaati aikaa niiden opetteluun ja testaamiseen hiekkalaatikossa. Lisäksi rinnakkaisten ohjelmistovaatimusten kohdalla työnkulku ei alusta asti ollut yksiselitteinen ja oikean toimintamallin löytäminen osoittautui työlääksi.

Rinnakkaisten ohjelmistoversioiden luominen ja niiden hallitseminen on monivaiheista ja työlästä. Laajemmissa projekteissa yksittäisiä dokumentteja voi olla kansioittain, ja pelkkä dokumenttien siirtäminen oikean ohjelmistoversion kansioon manuaalisesti lisää inhimillisten virheiden riskiä. Lisäksi Baselinen luominen jokaiselle yksittäiselle dokumentille kerrallaan on erittäin aikaa vievää.

Myös muiden inhimillisten riskien mahdollisuus on olemassa. Esimerkiksi tilanteessa, jossa valmis rinnakkainen osuus on jo mergetty takaisin Mainlineen,

mutta uuden version dokumentteja ei ole vielä ehditty jäädyttää, voi syntyä ongelmia. Toinen käyttäjä voi vahingossa lisätä uutta sisältöä haarautettuun dokumenttiin, jolloin jokin rinnakkainen versio eteneekin Mainlinea pidemmälle. Näin ei saisi tapahtua, ja siksi olisikin tärkeää noudattaa edellä mainittua työnkulkua vaihe vaiheelta.

Lisäksi jäljitettävyyden eheyden kanssa havaittiin haasteita. Esimerkkitapauksessa käyttäjä tekee muutoksia korkean tason vaatimukseen, johon on linkitetty matalan tason vaatimus. Tallennuksen jälkeen Polarion luo uuden revision korkean tason vaatimukselle, ja matalan tason vaatimuksen linkitys osoittaa nyt väärään revisioon, mikä sotkee jäljitettävyyden. Tämän vuoksi korkean tason vaatimukset tulisi aina hyväksyä ennen matalan tason vaatimuksia.

7 YHTEENVETO

Opinnäytetyössä perehdyttiin vaatimustenhallinnan käsitteisiin ja tutkittiin rinnakkaisten ohjelmistoversioiden vaatimustenhallintaa Siemensin Polarion ALM -työkalun avulla. Opinnäytetyön käytännönosuudessa laadittiin toimeksiantajan toiveesta Polarionin käyttäjille opas helpottamaan vaatimustenhallintaa.

Vaatimustenhallinnan avulla varmistetaan, että ohjelmiston toiminnalliset ja tekniset tarpeet täyttyvät. Hyvin hallitut vaatimukset johtavat parempaan projektisuunnitteluun, resurssien tehokkaampaan käyttöön sekä vähentävät merkittävästi ohjelmiston virheiden ja turvallisuusriskien määrää. Erityisen tärkeää tämä on ilmailualalla, jossa siviili-ilmailun DO-178C-standardi asettaa tiukat vaatimukset ohjelmistokehityksen eri vaiheille.

Rinnakkaisten ohjelmistoversioiden hallinta tuo omat haasteensa, sillä samanaikaisesti kehitettävät versiot on testattava, dokumentoitava ja hallittava erikseen. Rinnakkainen kehitys lisääkin usein merkittävästi tarvittavia resursseja. Haastavinta prosessissa on dokumentaation huolellinen ylläpito, jotta eri versioiden kehitys ja vaatimusten täyttyminen ovat selkeästi jäljitettävissä.

Polarion tarjoaa kattavat työkalut versionhallintaan ja jäljitettävyyden varmistamiseen. Siksi se on toimiva ratkaisu DO-178C-standardin asettamiin vaatimuksiin. Polarionin avulla on mahdollista seurata ja dokumentoida kaikkia ohjelmistokehityksen vaiheita, mikä vähentää virheiden riskiä ja varmistaa, että vaatimukset täyttyvät jokaisessa ohjelmistoversiossa. Syvemmin ohjelmaan perehdyttäessä kuitenkin havaittiin, että monista ohjelman vahvuuksista huolimatta, rinnakkaisten ohjelmistoversioiden hallinta ei ole yksinkertaista Polarionissakaan. Se vaatii monien eri vaiheiden oppimista ja toistuvien työvaiheiden hallitsemista. Lisäksi virheiden tekeminen on edelleen mahdollista, ja osa virheistä voi jäädä käyttäjältä huomaamatta.

Työkalun käyttö vaatii perehtymistä ja huolellisuutta. Kehitysehdotuksena yritykselle voisi olla lisäkoulutusten tarjoaminen työntekijöille, jotta Polarionin käyttö

saataisiin mahdollisimman sujuvaksi ja virheiden määrä minimoitua. Lisäksi jatkokehityksenä voitaisiin tutkia, kuinka prosesseja saataisiin automatisoitua enemmän. Dokumentointiin ja versionhallintaan liittyvä manuaalinen työ altistaa aina inhimillisille virheille.

Työn käytännönosuudessa laadittu opas tarjoaa selkeät ohjeet Polarionin käyttöön. Oppaan avulla pyritään varmistamaan, että myös uudet käyttäjät oppivat hallitsemaan eri ohjelmistoversioiden vaatimukset tehokkaasti ja oikein. Opas toivottavasti auttaa erityisesti työntekijöitä, joilla on vähemmän kokemusta Polarionin käytöstä sekä parantaa yleisesti sovelluksen hyödyntämistä yrityksen ohjelmistokehitysprojekteissa.

Toimeksiantajalle laadittu opas on työkalu, joka tukee yrityksen tavoitteita tehostaa työntekijöiden osaamista ja vähentää ohjelmiston kehitysprosessiin liittyviä virheitä. Se auttaa työntekijöitä noudattamaan DO-178C-standardin mukaisia toimintamalleja. Opas on hyödynnettävissä myös muihin yrityksen projekteihin, jotka käyttävät Polarionia rinnakkaisten ohjelmistoversioiden vaatimustenhallinnassa.

LÄHTEET

AVIORED. 2023. DO-178C Software Development for Safety Critical Systems. Koulutusmateriaali. Viitattu 21.7.2024. Vaatii käyttöoikeuden.

AVIORED. 2023. Systems Engineering. Koulutusmateriaali. Viitattu 21.7.2024. Vaatii käyttöoikeuden.

Binary Terms. 2024. Requirement Engineering. Verkkosivu. Viitattu 18.9.2024. https://binaryterms.com/requirement-engineering.html

Built In. 2023. What Is the V-Model in Software Development? Verkkosivu. Viitattu 5.11.2024. https://builtin.com/software-engineering-perspectives/v-model

GeeksforGeeks. 2024. Functional vs Non-Functional Requirements. Verkkosivu. Viitattu 18.9.2024. https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/

Paakki, J. 2011. Ohjelmistojen vaatimusmäärittely. Helsingin Yliopisto: Luentomateriaali. Viitattu 18.9.2024. https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf

POLARION. 2023. Administrator and User Help, Software Version Polarion 2310. Pdf-dokumentti. Viitattu 16.6.2024. Vaatii käyttöoikeuden.

ResearchGate. 2017. DO178C Software Life Cycle Processes. Verkkosivu. Viitattu 23.7.2024. https://www.researchgate.net/figure/DO178C-Software-Life-Cycle-Processes_fig1_318259306

ResearchGate. 2021. A Requirements Management Template in Polarion for Model-Based Development of Airborne Systems. Verkkosiu. Viitattu 5.11.2024. https://www.researchgate.net/figure/Requirement-Work-Item-Workflow-with-Optional-Extension_fig1_358646267

Siemens Polarion. 2024. Polarion® ALM™. Verkkosivu. Viitattu 30.8.2024. https://polarion.plm.automation.siemens.com/products/polarion-alm

Sommerville, I. 2011. Software Engineering, 9th Edition. Boston: Pearson Education, Inc.

Taipuva Consulting Oy. 2024. Polarion ALM. Verkkosivu. Viitattu 30.8.2024. https://www.taipuva.com/digital-solutions/polarion/

TopScaler. 2023. Difference between High-Level and Low-Level Design. Verkkosivu. Viitattu 18.9.2024. https://www.scaler.com/topics/high-level-design-and-low-level-design/

Vehkalampi, L. 2024. Document handling and lifecycle during software development and release process. Vaatii käyttöoikeuden.