Tampereen yliopisto

Ossi Kronlöf

KEHITYSKÄYTÄNTÖJEN PARANTAMINEN MIKROTIIMISSÄ

Ketterän kehitystavan ja versionhallinnan käyttöönotto ohjelmistoprojektissa

Diplomityö
Tietotekniikan tiedekunta
Tarkastajat: Prof. Outi Sievi-Korte
Kesäkuu 2022

TIIVISTELMÄ

Ossi Kronlöf: Kehityskäytäntöjen parantaminen mikrotiimissä
Diplomityö
Tampereen yliopisto
Ohjelmistotuotanto
Kesäkuu 2022

Diplomityössä tutkitaan ohjelmistokehitysprojektia mikrotiimissä konsultin näkökulmasta. Tutkimus on toteutettu tapaustutkimuksena SQM-Toimi -ohjelmiston uuden version kehitysprojektista. Kehitettävää ohjelmistoa ja organisaatiota sen ympärillä tutkitaan alkutilan ja projektin aikana tehtyjen toimien tarkastelun kautta. Projektin alussa havaittiin haasteita teknisen toteutuksen kanssa erityisesti ohjelmistotuotannon periaatteiden sekä nykyaikaisten kehitystyökalujen käytön osalta. Lisäksi osaamisen kehittämisen tarvetta havaittiin kehityskäytännöissä ja kehitysprosessissa.

Diplomityö tutkii nykyaikaisten kehityskäytäntöjen ja työkalujen tuontia projektiin kaksijakoisesti. Työkalujen valintaa, käyttöönottoa ja koulutusta tutkitaan yhtenä puolena kehitystä. Tehtyjä valintoja ja toimintaa tarkastellaan verraten ohjelmistotuotannon ja tiedonhallinnan kirjallisuuteen. Osaltaan työkaluvalinnat voidaan todeta hyväksi kirjallisuuteen verraten, mutta osa niistä on tehty kevyin perustein. Vaikutusten tarkastelu pitkällä aikavälillä empiirisesti on rajattu pois rajallisen aikaikkunan asettamien rajoitteiden takia. Ajallinen rajaus vaikuttaa myös kehitettyjen käytäntöjen pysyvyyteen, ylläpitoon ja jatkuvaan oppimiseen negatiivisesti ja estää niiden empiirisen evaluoinnin.

Toinen puoli tutkimuksen käsittelyosaa lähestyy projektia tiedon ja osaamisen hallinnan näkökulmasta. Kehityskäytäntöjen tuomista projektiin tarkastellaan tiedonhallinnan prosessin kautta, josta tietotarpeiden määritys ja tiedon analyysi on nostettu tarkempaan tarkasteluun. Konsultin tekemää koulutusta ja ohjausta tarkastellaan verraten osaamisen kehittämisen kirjallisuuteen.

Kehityskäytäntöihin ja työn struktuuriin tehtiin projektissa merkittäviä muutoksia. Tämä avaa tutkimukselle myös muutosjohtamisen näkökulman. Osaamisen kehittämisellä on organisaation oppimisen kautta vahva kytkös muutosjohtamisen aiheisiin, ja näitä käsitellään osaamisen hallinnan ohessa. Lisäksi projektin ja organisaation muutosjohtamista tarkastellaan alan kirjallisuuteen verraten keskustelun muodossa.

Tuloksena tutkimuksessa käsiteltävässä projektissa on sekä ohjelmistotuotannon että tietojohtamisen alan kirjallisuudessa havaittuja piirteitä. Konsultin rooli ohjautui projektissa odotettua vähemmän tekniseen konsultointiin ja enemmän johtavaan rooliin. Havainto tukee konsulttien laajempialaista kouluttamista eri aihealueista pelkän teknologiaosaamisen sijaan. Tuloksena huomataan myös tiedonhallinnan prosessin tärkeys tietojohtamisen kirjallisuuden tukemana. Suuremman resurssin varaaminen tiedonhallinnan prosessiin projektin alkuvaiheessa auttaa kohdistamaan resursseja projektin kuluessa ja mahdollistaa kehityskohteiden arvottamisen ja rajauksen.

Avainsanat: osaamisen hallinta, ohjelmistotuotanto, mikrotiimi, konsultointi

ABSTRACT

Ossi Kronlöf: Improving development practices in a microteam
Master of science thesis
Tampere University
Software engineering
June 2022

This master's thesis analyzes a software development project in a microteam from the perspective of an outside consultant. The research is a case study of the development project of a new version of the SQM-Toimi software. The developed software and the organization surrounding its development are reviewed through defining the initial state of the project organization and analyzing changes made during the project. In its initial state, the project suffered from challenges with technical implementation, especially with the lack of modern software engineering principles and development tools. Additionally, a lack of skill and expertise was found in best practices for development and software development processes.

The analysis section of the research is divided into two partially distinct topics. Selection, introduction and training of new tools during the project is reviewed as one side. The choices made and the actions taken are reviewed in relation to software engineering and knowledge management literature. In part, the selection of development tools can be seen as successful based on literature, but some of them were made without enough informational foundation. Analyzing long-term results empirically has been deemed out of the scope of this research due to restrictions set by a limited time window. Limited time also restricts actions to ensure stickability, maintainability and continual improvement and makes empirical evaluation of these aspects impossible.

The other side of the analysis takes a look at the project from an information and skill management perspective. The establishment of modern software development practices in the project organization is reviewed through the information management process model with a focus on defining information needs and information analysis. Work coordination and teaching undertaken by the consultant are reviewed by comparison to skill development and organizational learning literature.

Software development practices and structuring of work underwent notable changes during the project. This enables the research to also take on a change management perspective. Skill development is strongly tied to change management through organizational learning topics, and leading change in the project is reviewed with skill management. Additionally, change management in the project and organization is reviewed with relevant literature in the form of discussion.

As a result, common elements from both software engineering and information management literature were found in the studied project. The role of the software consultant in the project was focused surprisingly little on technical consulting and more on management. This finding supports training consultants in a wider array of topics rather than only technical expertise. As another result, the importance of a knowledge management process is highlighted, a finding supported by information management literature. Frontloading more resources to the knowledge management process sees the benefit of better focusing of resources and facilitates better priorization and scoping of development.

Keywords: skill management, software engineering, microteam, consulting

ALKUSANAT

Kirjoitin tämän diplomityön todella kovassa kiireessä henkilökohtaisen elämän tultua tutkimuksen ja työelämän tielle. Kandidaatintyöni tapaan tukenani olivat rakas avovaimoni Taru ja kissamme Mauri. Kiitokset menevät myös äidille ja isälle avautumisen vastaanottajina puhelimitse.

Kiitos kuuluu tietenkin myös asiakkaallemme SQC Oy:lle ja heidän ihanille työntekijöilleen. Ilman teitä tämä ei olisi ollut lainkaan mahdollista, ja tämä on ollut tähänastisen työurani kiinnostavin projekti.

Eindhovenissa, 2. kesäkuuta 2022

Ossi Kronlöf

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

Git	Versionhallinta (Version Control System, VCS)
Html	Hypertext Markup Language. Yleisesti verkkokehityksessä käytetty merkintäkieli.
PHP	Enimmäkseen verkkosovellusten käyttöliittymissä hyödynnettävä ohjelmointikieli.
Repositorio	Gitin tapa mallintaa tietoa.
Wiki	Wiki on verkkosivusto, jonka sisältöä käyttäjät voivat itse muokata haluamallaan tavalla.

1. JOHDANTO

Diplomityö tarkastelee ohjelmistoprojektia, jossa kehitettiin uusi versio asiakasyrityksen SQM-Toimi -ohjelmistosta. SQM-Toimi ohjelmiston tehtävänä on sosiaali- ja terveysalan asiakastietojen hallinta ja raportointi. Ohjelmistoprojektin kulkua tarkastellaan konsultin näkökulmasta. Pääasiallisena tehtävänäni projektissa oli asiakkaan kehittäjän avustaminen, koulutus ja ohjaus.

Asiakkaalla ei ollut projektin alkaessa käytössä nykyaikaisia kehityskäytäntöjä. Projektissa otettiin asteittain käyttöön versionhallinta, ketterä työtapa sekä useita työtehtävien jäsentelyyn, tiedon ja osaamisen hallintaan, sekä ohjelmakoodin laadun parantamiseen tähtääviä työkaluja ja käytäntöjä. Tutkimus syventyy tarkastelemaan näiden implementointia tietojohtamisen ja muutosjohtamisen kehyksessä ja vertaamaan alan kirjallisuutta tehtyihin valintoihin. Tutkimuskysymyksenä on: Miten tuoda nykyaikaisia kehityskäytäntöjä ohjelmistoprojektiin, jossa kehitystä on jo tehty mutta käytännöt puuttuvat?

Tutkimuskysymyksen tukena vastataan alikysymyksiin: Millä perusteilla käyttöön otettavia työkaluja kannattaa valita? Miten määrittää tietotarpeet ja käyttää tiedonhallinnan prosessia teknisen sekä osaamisen kehittämisen konsultoinnin tukena?

Tutkimus tarjoaa ratkaisuja ohjelmistoprojekteihin, jossa ohjelmistoa on jo kehitetty, mutta nykyaikaiset kehityskäytännöt puuttuvat. Kokonaisuus rakentuu kahdesta eri osa-alueesta: kehitystyökalujen valinnasta ja käyttöönotosta sekä tiedon ja osaamisen hallinnasta. Tutkimusmenetelmäksi osa-alueiden tarkasteluun on valittu tapaustutkimus. Tarkasteltavien käytäntöjen matala lähtötaso ja merkittävä kehitys projektin aikana luo erinomaisen pohjan analysoida implementoinnin kehitystä ja tuloksia.

Luvussa kaksi käsitellään projektityöskentelyn ja siihen liittyvien johtamistapojen teoreettista taustaa. Lisäksi luku esittelee tutkittavassa projektissa hyödynnetyt kehityskäytännöt ja -työkalut.

Luvussa kolme tarkastellaan kehitysprojektin alkutilaa. SQM-Toimi -ohjelmiston edellisen version kehityksestä edetään haasteille ja perustellaan niiden kautta motivaatio muutoksiin projektissa.

Luku neljä etenee käsittelemään kehitystyökalujen valintaa ja niiden tuontia projektiin. Luvussa viisi vastaavasti käsitellään tiedon ja osaamisen hallintaa ja kehitystä projektissa.

Luvussa kuusi analysoidaan saatuja tuloksia keskustelun muodossa ja tarkastellaan havaittuja rajoitteita.

2. TEOREETTISTA TAUSTAA TIEDON JA OSAAMISEN HALLINNASTA OHJELMISTOPROJEKTIN KONTEKSTISSA

2.1 Projektityöskentely

Projektiluontoinen työskentely on muodostunut tietotyössä oletusarvoksi [1]. Ohjelmistoala on verrattain uusi liiketoiminta-alue, minkä johdosta projektimallia on hyödynnetty alusta alkaen. *A guide to the project management body of knowledge* määrittelee projektin "väliaikaiseksi pyrkimykseksi ainutlaatuisen tuotteen, palvelun tai tuloksen luomiseksi". Projektilla on alku ja loppu, ja projektit voivat olla itsenäisiä tai toimia osana ohjelmaa tai portfoliota. [1]

Projektit toimivat tapana luoda arvoa organisaation käytössä olevista resursseista [1]. Tämä tekee resurssien kohdistamisesta tärkeän tekijän projektin onnistumisessa. Tietotyössä erityisesti henkilöstön osaamisen rooli kohdistettavana resurssina korostuu.[49] Nykyaikainen henkilöstöjohtaminen ei rajoitu näkemään henkilöstöä pelkästään resursseina. Useimmissa menestyksekkäissä projekteissa projektipäälliköillä on vahva emotionaalinen kyvykkyys, jonka tärkeimpiä tekijöitä ovat herkkyys (sensitivity), vaikutuskyky (influence), motivaatio (motivation) ja tunnollisuus (conscientiousness). [36] Resurssien kohdentaminen vaikuttaa projektityön tehokkuuteen. Työtehtävien vaativuuden kasvaessa osaamisen hallinnan rooli korostuu, kun työn tehokkuus muuttuu yhä vahvemmin riippuvaiseksi sen suorittajan osaamisesta.

Projektin johtaminen on ihmisten johtamisen lisäksi tiedon johtamista. Onnistunut tiedonhallinta projektin johtamisessa mahdollistaa tehokkaan ja tietoperustaisen päätöksenteon taktisella tasolla. Projektityöntekijöiden kannalta tiedonhallinta toimii arvon yhteisluonnin mahdollistajana ja parantaa työn tehokkuutta: työtehtävien päällekkäisyys vähenee, operatiivinen päätöksenteko helpottuu, ja tiedonhankintaan kuluu vähemmän aikaa. [50, 20]

2.1.1 Osaamisen hallinta

Osaamisen hallinta on kehittynyt yksilöiden osaamisten summaamisesta holistiseen näkemykseen organisaation osaamisesta. Projektityöskentelyn kontekstissa voidaan tar-

kastella projektiorganisaatiota ja sen osaamista. Osaamisen arviointi ja mittaus toimii työkaluna (projekti-)organisaation nykytarpeisiin vastaamiseen. Vähintään yhtä tärkeänä nähdään osaamisen jatkuva kehittäminen. Osaamisen kehittämisen kautta osaamisen hallintaa yhdistetään organisaation oppimisen teemoihin. [39]

Alkuperäinen viitekehys organisaation oppimisen tarkasteluun on yksilöoppimisen teoria (individual learning theory). Yksilöoppimisen teoria näkee organisaation oppimisen koostuvan yksilöiden mentaalimallien kehityksestä. Mentaalimallien kehitys tapahtuu yksilöiden saadessa lisää tietoa ja tietämystä. Erillisten yksilöiden mentaalimallien kehitys johtaa teorian mukaan yhteisvaikutuksena sekä yksilöiden että organisaation tiedonkäsittelyn tehostumiseen ja parempaan päätöksentekoon. [19]

Näkemys organisaation oppimisesta on kehittynyt sosiaalisen oppimisen teorian yleistyttyä alan kirjallisuudessa. Sosiaalisen oppimisen teoria haastaa yksilöoppimisen näkemyksen yksilön mielen pääosasta oppimisessa. Yksilöoppimisen teoria väittää oppimisen alkavan ajatusmallien muutoksesta, mikä nähdään ristiriitaisena kyvylle oppia asioita harjoittelemalla. Asioiden oppiminen harjoittelemalla kehittää ajatusmalleja tekemisen seurauksena, eikä ajatusmallien muutos näin voi olla esivaatimus oppimiselle. [15] Nykyaikainen organisaation oppimisen kirjallisuus käyttää valtaosaksi sosiaalisen oppimisen viitekehystä. Oppimisen nähdään olevan osa jokapäiväistä organisaation toimintaa. Oppiminen tapahtuu osana osallistumis- ja vuorovaikutusprosesseja. [19]

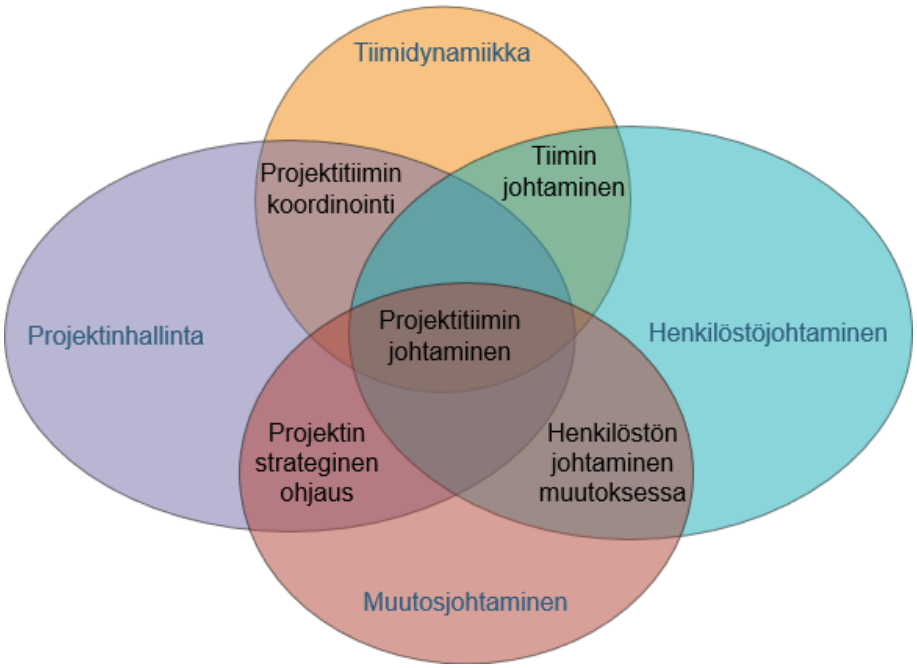
2.1.2 Projektitiimin johtaminen

Projektitiimin johtaminen on yhdistelmä useampaa eri johtamistieteiden aluetta. Muun muassa tiimidynamiikka, muutosjohtaminen, henkilöstöjohtaminen ja projektinhallinta tarjoavat käytäntöjä ja toimintamalleja projektitiimin johtamiseen. Kuvassa 2.1 on esitetty Venn-diagrammi projektitiimin johtamiseen liittyvistä osa-alueista.

Tiimin johtaminen

Projektiluontoisen työskentelyn tapaan tiimityöskentelyn suosio organisaatioissa on kasvanut merkittävästi vuosien mittaan. Tiimeihin perustuvilla rakenteilla organisaatiot pyrkivät vastaamaan toimintaympäristönsä kasvavaan monimutkaisuuteen. Parhaimmillaan tiimityöskentely mahdollistaa kattavampia ja innovatiivisempia ratkaisuja organisaation haasteisiin kuin itsenäinen työskentely. [44] Sosiaalisen oppimisen teorian kautta tästä voidaan luoda kytkös myös organisaation oppimisen kirjallisuuteen. [19].

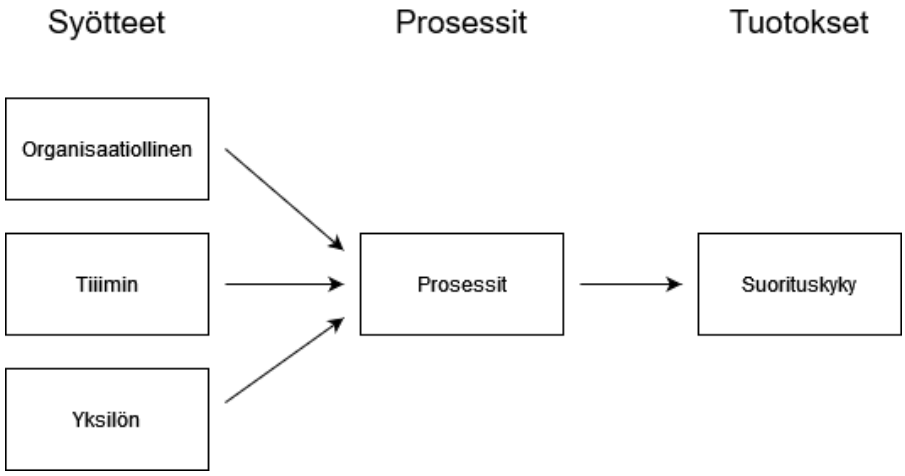
Tiimityöskentely ei tapahdu itsestään. Edes hyvä tiimirakenne ja monipuolinen yksilöosaaminen tiimissä eivät takaa tiimin menestystä. [23] Tiimit tarvitsevat menestyäkseen tiimityötä (teamwork). Tärkeänä tiimityön osa-alueena on tiimin johtaminen. Tiimin johtaminen on tiedolla johtamista, ja siihen sisältyy tiimin jäsenten tehtävien ohjaus ja koordi-



Kuva 2.1. *Venn-diagrammi projektitiimin johtamisen osa-alueista*

nointi, tehtävien jako, tiimin motivointi ja positiivisen ilmapiirin luominen. Tiimin johtaminen on myös tiedon johtamista: tiimin suorituskyvyn arviointia, suunnittelua ja organisointia, sekä tiimin tietämyksen, taitojen ja kykyjen kehittämistä. [44, 50]

Tiimin johtamisella pyritään parantamaan tiimien tehokkuutta. Tehokkuutta voidaan mitata esimerkiksi input-process-outcome (IPO) -mallilla. IPO-malli esittää, kuinka eri syötteistä voidaan tuottaa prosessien kautta tuotoksena suorituskykyä. Mitä enemmän suorituskykyä tuotetaan suhteessa syötteisiin, sitä tehokkaammin tiimi toimii. Malli on ollut käytössä tiimityöskentelyn tutkimuksessa jo lähes 60 vuotta, ja sitä on vuosien varrella kehitetty eteenpäin askeleittain. [34] Malli on esitetty kuvassa 2.2.



Kuva 2.2. *IPO-malli (suomennettu) [34]*

Tiimien epäonnistumisella voi olla merkittäviä vaikutuksia koko organisaatioon. Tiimin joh-

tamisen tavoitteena on tehokkuuden kasvattamisen lisäksi epäonnistumisten vähentäminen sekä niiden vaikutusten hallinta. [44] Hyvin johdetussa tiimissä epäonnistumisia voidaan hyödyntää organisaation oppimisen työkaluna [2]. Tarkoituksellista epäonnistumista hyödynnetään innovatiivisuudestaan tunnetuissa pienyrityksissä, kuten peliyritys Supercellillä [40].

Palaute

Palautetta voidaan hyödyntää projektitiimin johtamisessa sekä osaamisen kehittämisen ja työtehtävien koordinoinnin, että tiimin motivoinnin työkaluna. Tiimin johtamisen näkökulmasta palaute voidaan nähdä kolmena osa-alueena [33]:

- palautteen antaminen
- palautteen vastaanottaminen
- tiimin palautekulttuurin kehittäminen

Palautteenantoon tarjotaan kognitiivisen psykologian ja johtamisopin kirjallisuudessa useita malleja. Yksi tunnetuimmista palautteenannon malleista on niin kutsuttu palautehampurilainen (feedback sandwich) [17]. Palautteenannon hampurilaismalli on kuitenkin haastettu alan modernissa kirjallisuudessa [17, 53, 28]. Mallin on osoitettu yleisesti aiheuttavan hämmennystä ja vähentävän luottamusta [53].

Palautteenannon kehysten lisäksi alan kirjallisuus tarjoaa keinoja palautekulttuurin kehittämiseen ja ylläpitoon työyhteisössä. Näihin sisältyvät nykyaikaiset palautteenantomallit, kuten *The Leader Lab* -teoksessa esitetty Q-BIQ. Q-BIQ -malli koostuu kysymyksestä (**Q**uestion), käytöksestä (**B**ehavior), vaikutuksesta (**I**mpact) ja kysymyksestä (**Q**uestion), luoden näin luontevan ja helposti seurattavan struktuurin palautteen jäsentelyyn. [33]

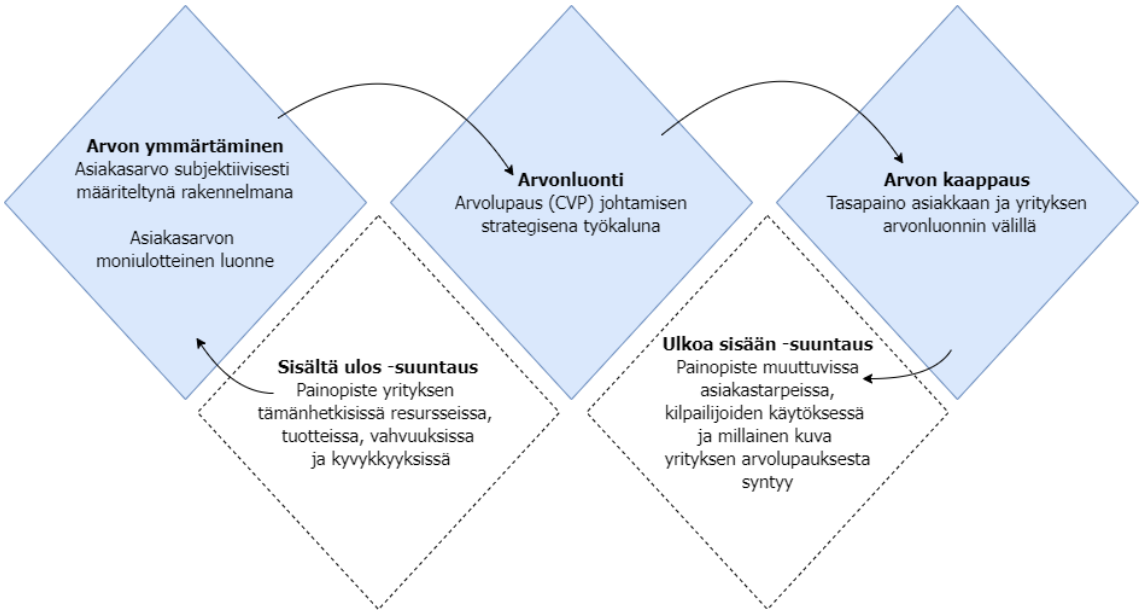
Toinen esimerkki nykyaikaisesta palautteenantomallista on *Feedback that works* -teoksessa esitelty SBI. Q-BIQ:in tavoin SBI-malli jakaa palautteenannon helposti seurattaviin askeliin. SBI kehottaa luomaan tilannekuvan (Capture the situation), kuvaamaan käytöksen (Describe the behavior) ja viimeisenä selittämään vaikutuksen (Explain the impact). [31]

Muutosjohtaminen

Projektityöskentelyssä on lähes poikkeuksetta kyse muutoksesta. Projektiorganisaation luominen on jo itsessään usein suuri muutos, kun työntekijöiden keskinäisiä yhteistyösuhteita järjestellään uudestaan, ja tiimityön tavoitteet ja struktuuri muotoillaan palvelemaan projektin tarpeita. Muutosjohtamisen näkökulma on tärkeä osa projektitiimin johtamista.

Johtajuudella (leadership) on merkittävä rooli muutoksen onnistumisessa [24]. Asiakasprojekteissa asiakkaalle tuotettu arvo on projektin ja muutoksen keskiössä. Arvon yhteisluonnin fasilitoimiseksi voidaan hyödyntää muun muassa asiakasarvokehystä (customer

value framework). Kehys korostaa ulkoa sisään -suuntauksen tärkeyttä arvonluontiprosessin tarkastelussa. [55] Asiakasarvokehys on esitelty kuvassa 2.3.



Kuva 2.3. *Asiakasarvokehys (kirjoittajan suomentama) [55]*

2.2 Aineettoman pääoman johtaminen ohjelmistoprojektissa

Aineettoman pääoman johtaminen on tietojohtamisen alue, joka keskittyy tiedon luontiin, kehitykseen ja siirtymiseen organisaatioissa. Aineeton pääoma voidaan jaotella suhdepääomaan, rakennepääomaan ja inhimilliseen pääomaan. Kuvassa 2.4 on kuvattu osa-alueiden sisältö.

Kuva 2.4. *Aineettoman pääoman osa-alueet [32]*

Aineettoman pääoman johtaminen on erityisen tärkeää ohjelmistoprojekteissa, joissa tietoa hyödynnetään ja tuotetaan suuria määriä. [50] Ohjelmistokehitys on tietointensiivistä

palvelutoimintaa, ja vaatii työntekijöiltään laajaa tietämystä sekä substanssista että työn kontekstista.

Aineetonta pääomaa voidaan johtaa tietojohtamisen kirjallisuuden esittämillä prosesseilla ja käytännöillä. Ohjelmistoprojektin kaltaisessa digitaalisessa ympäristössä tiedonhallintaan hyödynnetään laajasti myös erilaisia ohjelmallisia alustoja ja työkaluja. Johtamisen rooli korostuu alustojen käyttöönotossa ja tehokkaassa hyödyntämisessä - tietojärjestelmä ei tehosta työtä ilman prosesseja sen ympärillä. [13]

2.2.1 Dokumentaatio

Dokumentaatio toimii työkaluna muuttaa hiljaista tietoa dokumentoiduksi tiedoksi. Dokumentoitua tietoa voidaan hyödyntää tiedon jalostamisessa tietämykseksi sekä tiedonsiirrossa. Dokumentoitu tieto on organisaation rakennepääomaa, joka ei myöskään ole inhimilliseksi pääomaksi luettavan hiljaisen tiedon tavoin vaarassa hävitä organisaatiosta henkilöstön vaihtuessa. [50]

Confluence wiki

Confluence wiki on Atlassianin kehittämä ja julkaisema yhteistoiminnallinen dokumentaatioalusta [3]. Rakenteeltaan Confluence vastaa tyypillistä wiki-sivustoa: tieto on jaettu artikkeleihin, joita voi sijoitella vapaasti kansiorakenteeseen ja viitata keskenään ristiin. Confluence toimii SaaS, eli software as a service -mallilla. Tämä tarkoittaa, että Atlassian tarjoaa asiakkaille valmiin ratkaisun, jossa toimittaja on vastuussa järjestelmän käyttöönotosta, ajoympäristöstä ja ylläpidosta. [30]

2.2.2 Kommunikaatiokäytännöt

Dokumentaation tavoin kommunikaatiokäytännöt ovat osa organisaation rakennepääomaa. Kommunikaatiokäytännöt rakentuvat kuvan 2.4 osa-alueista arvojen ja kulttuurin sekä prosessien ja järjestelmien päälle. Hyvän kommunikaation toteutumiseen vaikuttaa kommunikaatiokäytäntöjen lisäksi myös organisaation inhimillinen pääoma, kuten osaaminen, henkilöominaisuudet, henkilökohtaiset verkostot ja asenne. [50]

Kommunikaatiokäytäntöjen kehittäminen tapahtuu tehokkaimmin organisaatiokulttuurin kautta. [24] Hyvä organisaatiokulttuuri tukee kommunikaatiokäytäntöjä tarjoamalla avoimen ja hyväksyvän ilmapiirin, jossa tiedon jakaminen on luontevaa ja kynnys siihen on matala. Lisäksi muutosmyönteinen organisaatiokulttuuri tarjoaa organisaation osille, kuten erilaisille tiimeille mahdollisuuden kehittää omia käytäntöjään ja tukee niiden ylläpitoa. [47]

Mattermost

Mattermost on verkkopohjainen vuorovaikutusalusta. Mattermost on Confluence wikin tavoin Atlassianin kehittämä järjestelmä. Se mahdollistaa nopean matalan kynnyksen kommunikaation sekä suoraan kahden käyttäjän välillä, että vapaasti määritettävillä keskustelukanavilla. [35]

2.3 Ketterä työtapa

Ketterä työtapa, eli Agile, on kokoelma metodeja ja käytäntöjä, jonka on tarkoitus vastata ohjelmistokehitysprojektien yleisimpiin haasteisiin. Näitä haasteita ovat esimerkiksi budjettiylitykset, myöhästyneet projektit, huonolaatuiset tuotokset ja pettyneet käyttäjät ja/tai asiakkaat. [16]

Ketterän kehityksen perusperiaatteet perustuvat vuonna 2001 julkaistuun *ketterän ohjelmistokehityksen julistukseen*. Julistus kuuluu kokonaisuudessaan näin:
Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:

- ***Yksilöitä ja kanssakäymistä*** *enemmän kuin menetelmiä ja työkaluja*
- ***Toimivaa ohjelmistoa*** *enemmän kuin kattavaa dokumentaatiota*
- ***Asiakasyhteistyötä*** *enemmän kuin sopimusneuvotteluja*
- ***Vastaamista muutokseen*** *enemmän kuin pitäytymistä suunnitelmassa [6]*

Ketterä kehitys pyrkii vähentämään tai poistamaan ylimääräiset häiriötekijät ja ongelmien lähteet kehitysprosessista ja ohjaamaan huomiota olennaisimpiin asioihin. Perinteisissä kehitysmalleissa suunnittelu on etupainotteista, mikä johtaa usein ongelmiin myöhemmissä vaiheissa kehitysprosessia. Ketterässä kehityksessä käytetään evolutiivista prosessia, joka on todettu tehokkaammaksi ja mukautuvammaksi. Evolutiivinen prosessi tarkoittaa, että ohjelmistoa kehitetään iteraatioissa. Perinteisessä vesiputousmallissa pyritään määrittelemään kehitystarpeet ennen toteutuksen aloittamista. Ketterässä kehityksessä määritetään ensin vähimmäistuote (MVP, Minimum Viable Product). Tätä vähimmäistuotetta ruvetaan kehittämään eteenpäin iteraatioina, joissa myös kehitystarpeita määritetään uudestaan toteutuksen kanssa limittäin. [46]

Scrum

Scrum on yksi ketterän ohjelmistokehityksen kehys. Scrum pyrkii ottamaan ketterän ohjelmistokehityksen periaatteet ja rakentamaan niiden pohjalta helposti sovellettavan ja tehokkaan prosessin ohjelmistokehitykseen. Scrum määrittää mm. kehitystiimin rakenteen, projektin etenemisen sekä tapoja etenemisen seuraamiseen. [45]

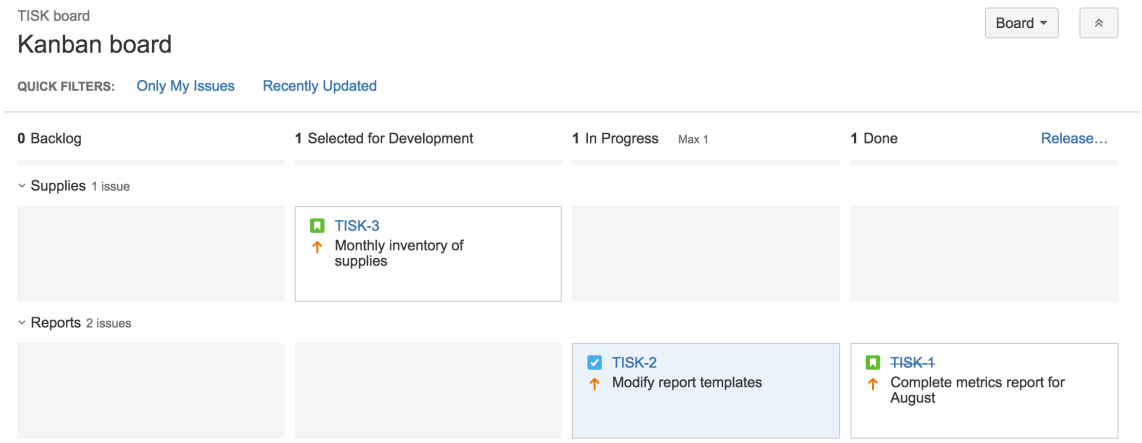
Scrumin tärkein lisämääritelmä ketterään kehitykseen on sprinttien käsite. Scrumissa työskentely tapahtuu (yleensä kahden viikon) jaksoissa. Sprinttien välissä tarkastellaan edellisen sprintin tuloksia retrospektiivissä ja määritetään seuraavan sprintin tehtävät ja tavoitteet sprint planningissa. Jaksojen aikana projektitiimi kokoontuu ideaalitilanteessa päivittäin hyvin lyhyeen palaveriin kertaamaan päivän tehtävänsä. [45]

Kanban

Kanban on hyvin paljon scrumin kaltainen kehitysmalli. Kanban on kuitenkin soveltamiseltaan joustavampi kuin Scrum. Kanban ei määrittele työtä tiettyihin ajanjaksoihin, vaan kehitystehtäviä voidaan siirtää backlogilta tehtävätaululle joustavasti tarvittaessa. [5]

Jira issue board

Jira issue board on verkkopohjainen tehtävätaulu. Myös Jira on Atlassianin julkaisema järjestelmä. Jira on suunnattu erityisesti ketterän kehityksen toimintatapaan, vaikka sitä voidaan soveltaa myös muihin ohjelmistokehitysmalleihin. Jiran päänäkymä on tehtävätaulu, johon voi määrittää yksittäisiä tehtäviä valitsemallaan tarkkuudella. Tehtävätaulun voi järjestellä eri tavoin, ja tehtäviä voi siirtää vetämällä haluamaansa järjestykseen. [4] Kanbanissa yleinen tehtävätaulun rakenne on esitetty kuvassa 2.5



Kuva 2.5. *Kanban-taulu Jira-järjestelmässä [5]*

2.4 Ohjelmakoodin laatu

Ohjelmakoodin laatu on laajasti tutkittu aihe [7, 41, 54, 48]. Ohjelmakoodin laadulla on havaittu olevan suora yhteys kilpailukykyyn ohjelmistokehitysalalla. Heikkolaatuinen ohjelmakoodi johtaa moniin erilaisiin haasteisiin ja ongelmiin, jotka voivat vaikuttaa niin kehittäjiin kuin loppukäyttäjiinkin. Huonolaatuinen ohjelmakoodi muun muassa kasvattaa todennäköisyyttä virheisiin, huonontaa ohjelmiston ylläpidettävyyttä ja laajennettavuutta sekä aiheuttaa teknistä velkaa. [43]

Versionhallinta

Versionhallinta (Version Control System: VCS) on nimensä mukaisesti järjestelmä kehitettävän tuotoksen hallintaan. Versionhallinta mahdollistaa yhtäaikaisen kehityksen useamman kehittäjän välillä, kehityksen haarauttamisen, edeltäviin versioihin palautumisen ja paljon muita ohjelmistokehityksessä jo lähes itsestäänselvyyksiksi muodostuneita toimintoja. Versionhallinnan on osoitettu nopeuttavan ja yksinkertaistavan ohjelmistokehitysprosessia. [56]

Git

Git on maailman käytetyin ratkaisu versionhallintaan. Git toimii erilaisella logiikalla kuin muut yleiset versionhallinnat, kuten *Apache Subversion* ja *Mercurial*. Yksittäisten tiedostojen ja niiden muutosten (patch) tallentamisen sijaan git tallentaa jokaisella tallennuksella (commit) koko hakemistopuun tilan. Jos tiedosto ei muutu tallennusten välissä, git tallentaa tiedoston sijaan linkin. [21]

3. TARKASTELTAVAN TAPAUKSEN ESITTELY

3.1 SQM-Toimi -ohjelmiston toteutus

SQM-Toimi on sosiaali- ja terveysalan asiakasraportointijärjestelmä. Sen ominaisuuksiin kuuluu muun muassa:

- Asiakasrekisteri
  - Raportointi
  - Käyttörahan seuranta
  - Laadunmittaus
- Tiedostonhallinta
- Ajanhallinta
- Muistutukset

SQM-Toimi on verkkopohjainen SaaS-ratkaisu. Asiakkaat käyttävät ohjelmistoa selaimen kautta heille osoitetussa osoitteessa, ja tiedonkäsittely tapahtuu palvelimella.

3.1.1 Group-Office

SQM-Toimi on toteutettu Group-Office -nimisen groupware-ratkaisun päälle. Ohjelmiston backend on PHP:tä, ja frontend JavaScriptiä ExtJS-kirjastolla varustettuna. Tietokanta on perinteinen SQL-relaatiokanta.

SQM-Toimi on toteutettu mukautettuina komponentteina Group-Officen pohjalta. Group-Officesta käytetään joitain valmiita näkymiä ohjelmiston tarpeisiin mukautettuna. Toiset näkymät taas on toteutettu lähes kokonaan itse.

3.1.2 Kehitystiimi

Projektia kehittää aktiivisesti vain yksi ohjelmistokehittäjä. Kehittäjä on opetellut itsenäisesti PHP-, html- ja JavaScript-kehitystä. Alan koulutusta kehittäjällä ei ole. Tekninen osaaminen löytyy, mutta ohjelmistokehityksen käytännöistä ei ole koulutuksen puutteen myötä juurikaan tietämystä tai osaamista.

3.1.3 Ohjelmointityyli

Ohjelmointityylissä havaitaan monia haasteita. JavaScriptin asynkronisuus koetaan monessa kohtaa haastavaksi. Ohjelmakoodi rikkoo useita kehityksen perusperiaatteita.

Useimmin rikottu periaate on DRY, eli *Don't Repeat Yourself*. Ohjelmakoodi sisältää paljon toisteisuutta. Tämä voi olla pienimmillään samankaltaisten ominaisuuksien toteuttamista eri moduuleihin, mutta pahimmillaan suurten määrien koodirivejä kopioimista useampaan paikkaan. Esimerkki periaatteen rikkomisesta voidaan nähdä kuvassa 3.1 lomakkeen toteutuksessa. Lomake sisältää 20 saman nimistä tekstikenttää, jotka on toteutettu kopioimalla sama ohjelmakoodi 20 kertaa. Koodileikkeessä nähdään ensimmäiset kahdeksan.

```
initFormItems: function () {
    this.customFields = go.customfields.CustomFields.getFormFieldSets("Careplan");
    this.fieldSet1 = new Ext.form.FieldSet({
        xtype: 'fieldset',
        items: [new go.modules.sqc.patientregistry.PatientCardCombo({value: go.modules.sqc.patientregistry.selectedPatient}),
            go.modules.sqc.patientregistry.text_1 = new go.form.HtmlEditor({
                name: 'text_1',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_2 = new go.form.HtmlEditor({
                name: 'text_2',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_3 = new go.form.HtmlEditor({
                name: 'text_3',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_4 = new go.form.HtmlEditor({
                name: 'text_4',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            })]
    });

    this.fieldSet2 = new Ext.form.FieldSet({
        xtype: 'fieldset',
        items: [
            go.modules.sqc.patientregistry.text_5 = new go.form.HtmlEditor({
                name: 'text_5',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_6 = new go.form.HtmlEditor({
                name: 'text_6',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_7 = new go.form.HtmlEditor({
                name: 'text_7',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            }),
            go.modules.sqc.patientregistry.text_8 = new go.form.HtmlEditor({
                name: 'text_8',
                fieldLabel: t("NotSaved"),
                height:170,
                anchor: '100%',
                grow: true,
            })]
    });
```

Kuva 3.1. *Esimerkki DRY-periaatteen rikkomisesta*

Koodileikkeessä kuvattua lomaketta voidaan käyttää esimerkkinä koodaustyylistä seuraavasta teknisestä velasta. Kuvissa 3.2 ja 3.3 on kuvattu lomakekenttien käsittely. Koska lomakekentät on luotu toisteisesti, eikä niitä ole tallennettu soveltuvaan tietorakenteeseen, myös niiden käsittely on päädytty tekemään hyvin toisteisesti ja suurella määrällä koodirivejä.

```
go.modules.sqc.patientregistry.getLabel(1,function(labels) {
    // set fieldLabels one by one
    // if subject is not saved, then item will be hided
    if(labels.subject_1) {
        go.modules.sqc.patientregistry.text_1.setFieldLabel(labels.subject_1);
    } else {
        go.modules.sqc.patientregistry.text_1.setVisible(false);
    }
    if(labels.subject_2) {
        go.modules.sqc.patientregistry.text_2.setFieldLabel(labels.subject_2);
    } else {
        go.modules.sqc.patientregistry.text_2.setVisible(false);
    }
    if(labels.subject_3) {
        go.modules.sqc.patientregistry.text_3.setFieldLabel(labels.subject_3);
    } else {
        go.modules.sqc.patientregistry.text_3.setVisible(false);
    }
    if(labels.subject_4) {
        go.modules.sqc.patientregistry.text_4.setFieldLabel(labels.subject_4);
    } else {
        go.modules.sqc.patientregistry.text_4.setVisible(false);
    }
                // second tab
                if(labels.subject_5) {
                    go.modules.sqc.patientregistry.text_5.setFieldLabel(labels.subject_5);
                } else {
                    go.modules.sqc.patientregistry.text_5.setVisible(false);
                }
                if(labels.subject_6) {
                    go.modules.sqc.patientregistry.text_6.setFieldLabel(labels.subject_6);
                } else {
                    go.modules.sqc.patientregistry.text_6.setVisible(false);
                }
                if(labels.subject_7) {
                    go.modules.sqc.patientregistry.text_7.setFieldLabel(labels.subject_7);
                } else {
                    go.modules.sqc.patientregistry.text_7.setVisible(false);
                }
                if(labels.subject_8) {
                    go.modules.sqc.patientregistry.text_8.setFieldLabel(labels.subject_8);
                } else {
                    go.modules.sqc.patientregistry.text_8.setVisible(false);
                }
    if(labels.subject_9) {
        go.modules.sqc.patientregistry.text_9.setFieldLabel(labels.subject_9);
    } else {
        go.modules.sqc.patientregistry.text_9.setVisible(false);
    }
    if(labels.subject_10) {
        go.modules.sqc.patientregistry.text_10.setFieldLabel(labels.subject_10);
    } else {
        go.modules.sqc.patientregistry.text_10.setVisible(false);
    }
    if(labels.subject_11) {
        go.modules.sqc.patientregistry.text_11.setFieldLabel(labels.subject_11);
    } else {
        go.modules.sqc.patientregistry.text_11.setVisible(false);
    }
    if(labels.subject_12) {
        go.modules.sqc.patientregistry.text_12.setFieldLabel(labels.subject_12);
    } else {
        go.modules.sqc.patientregistry.text_12.setVisible(false);
    }
    if(labels.subject_13) {
        go.modules.sqc.patientregistry.text_13.setFieldLabel(labels.subject_13);
```

Kuva 3.2. *Lomakekenttien käsittely, DRY-periaatteen rikkomisesta seuraava tekninen velka*

```
if(labels.subject_14) {
    go.modules.sqc.patientregistry.text_14.setFieldLabel(labels.subject_14);
} else {
    go.modules.sqc.patientregistry.text_14.setVisible(false);
}
if(labels.subject_15) {
    go.modules.sqc.patientregistry.text_15.setFieldLabel(labels.subject_15);
} else {
    go.modules.sqc.patientregistry.text_15.setVisible(false);
}
if(labels.subject_16) {
    go.modules.sqc.patientregistry.text_16.setFieldLabel(labels.subject_16);
} else {
    go.modules.sqc.patientregistry.text_16.setVisible(false);
}
if(labels.subject_17) {
    go.modules.sqc.patientregistry.text_17.setFieldLabel(labels.subject_17);
} else {
    go.modules.sqc.patientregistry.text_17.setVisible(false);
}
if(labels.subject_18) {
    go.modules.sqc.patientregistry.text_18.setFieldLabel(labels.subject_18);
} else {
    go.modules.sqc.patientregistry.text_18.setVisible(false);
}
if(labels.subject_19) {
    go.modules.sqc.patientregistry.text_19.setFieldLabel(labels.subject_19);
} else {
    go.modules.sqc.patientregistry.text_19.setVisible(false);
}
if(labels.subject_20) {
    go.modules.sqc.patientregistry.text_20.setFieldLabel(labels.subject_20);
} else {
    go.modules.sqc.patientregistry.text_20.setVisible(false);
}

if(!go.modules.sqc.patientregistry.text_1.isVisible() && !go.modules.sqc.patientregistry.text_2.isVisible() && !go.modules.sqc.patientregistry.text_3.isVisible() && !go.modules.sqc.patientregistry.text_4.isVisible())
{
    // if all items are invisible, remove the tab
    Ext.getCmp('careplan-tabpanel').remove('careplan-tab-1');
}

if(!go.modules.sqc.patientregistry.text_5.isVisible() && !go.modules.sqc.patientregistry.text_6.isVisible() && !go.modules.sqc.patientregistry.text_7.isVisible() && !go.modules.sqc.patientregistry.text_8.isVisible())
{
    // if all items are invisible, remove the tab
    Ext.getCmp('careplan-tabpanel').remove('careplan-tab-2');
}

if(!go.modules.sqc.patientregistry.text_9.isVisible() && !go.modules.sqc.patientregistry.text_10.isVisible() && !go.modules.sqc.patientregistry.text_11.isVisible() && !go.modules.sqc.patientregistry.text_12.isVisible())
{
    // if all items are invisible, remove the tab
    Ext.getCmp('careplan-tabpanel').remove('careplan-tab-3');
}

if(!go.modules.sqc.patientregistry.text_13.isVisible() && !go.modules.sqc.patientregistry.text_14.isVisible() && !go.modules.sqc.patientregistry.text_15.isVisible() && !go.modules.sqc.patientregistry.text_16.isVisible())
{
    // if all items are invisible, remove the tab
    Ext.getCmp('careplan-tabpanel').remove('careplan-tab-4');
}

if(!go.modules.sqc.patientregistry.text_17.isVisible() && !go.modules.sqc.patientregistry.text_18.isVisible() && !go.modules.sqc.patientregistry.text_19.isVisible() && !go.modules.sqc.patientregistry.text_20.isVisible())
{
    // if all items are invisible, remove the tab
    Ext.getCmp('careplan-tabpanel').remove('careplan-tab-5');
}
```

Kuva 3.3. *Teknisen velan kertautuminen ohjelmakoodissa*

Toinen selkeä haaste on KISS: *Keep It Simple Stupid*. Ilman koodikatselmointia ja tiimiä kehittämisen tukena, useat ominaisuudet on toteutettu vain ensimmäisellä tavalla, jolla ne on saatu toimimaan. Tämä tarkoittaa monin paikoin virhealtista ja vaikeasti laajennettavaa ja ylläpidettävää koodia. Tämän periaatteen rikkominen aiheuttaa myös järjestelmään tutustuvalle kehittäjälle haasteita ymmärtää ohjelmakoodin toimintaa.

Kun jokin ratkaisu ongelmaan löytyy, sitä on myös saatettu käyttää harkitsemattomasti muidenkin ongelmien ratkomiseen. Laajan tietämyksen puute vaihtoehtoisista ratkaisuista pahentaa tilannetta. Tällaista toimintatapaa kutsutaan *kultaiseksi vasaraksi*, jossa samaa työkalua pyritään käyttämään kaikkiin ongelmiin, vaikka parempia ratkaisuja olisi olemassa. Ohjelmakoodissa kultainen vasara voidaan nähdä esimerkiksi kuvassa 3.4. JavaScript-pohjaiseen näkymään on toteutettu toimintalogiikkaa, joka olisi toiminut yksinkertaisemmin ja varmemmin PHP-backendissä.

```
getCustomData: function (v) {
    // not the best possible solution, but it seems to work
    // any suggestions how to implement this better?
    customFields(v,function(html){document.getElementById("cf"+v).innerHTML=html}); // update innerHTML
    return ""; // return empty stings instead of 'undefined'

},
/*getCustomData: function (dbid) {
    // Server side solution, incomplete
    go.Jmap.request({
        method: "sqc/patientregistry/Careplan/customfieldsdata", // modules\sqc\patientregistry\controller\Careplan.php public function customfieldsdata($dbid)
        params: {
            id: dbid
        },
        callback: function(options, success, result) {
            go.modules.sqc.patientregistry.data=result.testi;
        },
        scope: this
    });

    return go.modules.sqc.patientregistry.data;
},*/
getDisplayName: function(name,userId,id) {
    var pr = new Promise(function (resolve, reject) {
        go.Db.store("User").single(userId).then(function(result) {
            var displayName=result.displayName;
            // same here, not the best possible solution, but it seems to work
            document.getElementById(name+id).innerHTML=displayName;
        });
    });
    return "";
}
```

Kuva 3.4. *Esimerkki kultaisesta vasarasta*

3.2 Kehitysprosessi

Tekninen toteutus on vain yksi osa ohjelmistokehitysprojektia. Toinen tärkeä osa-alue on kehitysprosessi. SQM-Toimi -projektissa kehitysprosessissa havaittiin useita osa-alueita, joiden lähtötila on tärkeää mainita.

3.2.1 Palvelinratkaisu

SQM-Toimen palvelinratkaisu on perinteinen. Palvelimet sijaitsevat alihankkijan konesalissa. Kehittäjä pystyy yhdistämään palvelimille SSH-yhteydellä, jonka kautta hän voi tehdä muutoksia tiedostoihin. Muuta pääsyä, kuten yleisiä komentorivitoimintoja, portin avauksia tai vastaavaa ei ole, vaan nämä pitää pyytää alihankkijalta tarvittaessa.

Projektin alussa SQM-Toimi -järjestelmässä oli varattu jokaiselle asiakkaalle oma palvelimensa. Palvelimella suoritettiin muusta ohjelmistosta riippumatonta asiakaskohtaista versiota ohjelmistosta. Koska ohjelmisto on asiakaskohtaisesti räätälöity, SQM-Toimi -järjestelmästä on yhtä monta koodipohjaa kuin asiakasta. Varsinaista seurantaa erillisten koodipohjien eroista ei ollut dokumentoitu.

3.2.2 Julkaisukäytännöt

Ohjelmiston julkaisut tehdään täysin käsityönä. Projektilla ei ole mitään julkaisuaikataulua tai kehityssykliä, vaan julkaisuja saatetaan tehdä lyhyellä varoitusajalla ja hyvin vaihtelevalle määrälle asiakkaita. Käytännössä uusi versio julkaistaan kopioimalla SSH-yhteyden yli uudet tiedostot palvelimelle ja uudelleenkäynnistämällä ohjelmisto. Isompien päivitysten yhteydessä tästä seuraa huomattavasti työtä, kun päivitykset tehdään jokaisen asiak-

kaan palvelimelle erikseen.

3.2.3 Struktuurin puute

Asiakaspalvelinten lisäksi ohjelmakoodi on tallennettuna ainoastaan kehittäjän omalle tietokoneelle. Tämän lisäksi testipalvelimilla on satunnainen kehitysversio, jonka tila riippuu tällä hetkellä kehitettävästä ominaisuudesta. Varmasti toimiva perusversio ohjelmistosta ei siis ole tallennettuna mihinkään hajautettuun tallennustilaan eikä siitä ole varmuuskopiota. Versionhallintaa ei ole, vaan eri asiakkaiden versiot ohjelmistoista ovat vain eri kansioissa kehittäjän tietokoneella.

Kehitysprosessia ei myöskään ole, ja kehitystehtävien seuranta tapahtuu ainoastaan palaverien kautta. Projektin dokumentaationa on ainoastaan asiakkaille suunnatut käyttöoppaat. Kehitystehtävät valitaan akuuttien asiakastarpeiden mukaan, työn ohjaamisen tueksi ei ole olemassa varsinaista prosessia.

3.3 Motivaatio muutoksiin

Edellä mainittu alkutila ei ole haluttava. Nykyaikaiset kehityskäytännöt puuttuvat lähes täysin. Ongelmia on sekä teknisessä toteutuksessa että prosesseissa.

3.3.1 Työmäärän vähentäminen

Merkittävä motivaatio lähteä muuttamaan kehitysprosessia on ylimääräisen työn vähentäminen. Lisätyötä aiheutuu ensinnäkin teknisestä toteutuksesta. Ohjelmistotuotannon periaatteita rikkovaa ohjelmakoodia on työlästä ylläpitää, ja se johtaa tekniseen velkaan. SQM-Toimen tapauksessa se on johtanut esimerkiksi samojen asioiden toteuttamiseen ja samojen ongelmien ratkaisemiseen moneen kertaan. Tämän tekijänä on myös dokumentaation puute - organisaation hiljaista tietoa on tarvetta muuttaa eksplisiittiseksi sen säilymisen, siirtymisen ja uudelleenkäytön parantamiseksi [50].

Myös struktuurin puute lisää ylimääräistä työtä. Heikko ymmärrys asiakastarpeista vaikeuttaa kehitystehtävien priorisointia. Suhteessa paljon työaikaa saatetaan käyttää arvoltaan matalaan ominaisuuteen, tärkeämpien ja vähempitöisten ominaisuuksien jäädessä toteuttamatta. Kehittäjän on myös vaikea keskittyä tekniseen toteutukseen, kun aikaa kuluu työtehtävien hallintaan ja määrittämiseen.

3.3.2 Työn laatu

Ohjelmakoodin vaihteleva laatu ei vaikuta ainoastaan työmäärään. Myös tuotteen laatu voi kärsiä huonolaatuisesta ohjelmakoodista [43]. Tämä voi johtaa odottamattomiin vir-

heisiin ja pahimmillaan heikkoon asiakaskokemukseen. Koodin laatu on myös tärkeää, jos projektiin tuodaan uusia työntekijöitä tai ulkoisia konsultteja - sen tulkitseminen on haastavaa.

Myös struktuurin puute vaikuttaa työn laatuun. Kun asiakastarpeita ja yrityksen visiota voidaan käyttää tehokkaammin kehityksen ohjaamiseen, myös tuotteen (tässä tapauksessa ohjelmiston) koettu laatu paranee - tuote tuottaa enemmän asiakasarvoa [26]. Lisäksi epäselvät työtehtävät, ylimääräiseksi koettu työ sekä vaikeasti havaittava edistyminen voivat vaikuttaa kehittäjien motivaatioon ja suorituskykyyn huomattavasti [14, 29]. Tämä myös johtaa osallaan huonompaan työn tulokseen: heikompilaatuiseen ohjelmistoon.

3.3.3 Resilienssi virheille ja onnettomuuksille

Alkutilassaan projekti on hyvin riskialtis. Virheitä voi päätyä tuotantoon asti katselmoinnin, testauksen ja laadunvarmistuksen puuttuessa. Havaittuja virheitä tai bugeja on usein vaikeaa selvittää ja korjata, koska versionhallintaa ei ole. On siis vaikeaa tietää, mitä osaa toteuttaessa havaittu virhe on syntynyt.

Suuremmalla mittakaavalla projektissa on riskinä peruskoodipohjan katoaminen. Näin voi tapahtua jos ohjelmistokehittäjän kiintolevy hajoaa. Myös niin kutsutut Force Majeuret, kuten tulipalot ja muut onnettomuudet voivat johtaa kriittiseen tiedonmenetykseen varmuuskopioiden puuttuessa. Nämä tilanteet eivät ole todennäköisiä, mutta niiden vaikutukset voivat olla katastrofaalisia. Kiintolevyjen tapauksessa mahdollisuus hajoamiseen kasvaa käyttöajan pidentyessä [25].

4. KEHITYSTYÖKALUJEN VALINTA JA TUONTI PROJEKTIIN

4.1 Versionhallinta

Tämän päivän ohjelmistoprojekteissa hyödynnetään lähes poikkeuksetta versionhallintaa. Selkeästi suosituin versionhallinta on Git. [21] Muitakin ratkaisuja, kuten Subversion, on tarjolla. Versionhallinta tarjoaa parannusta useampaan kehityksessä havaituista haasteista. Hajautettu versionhallintaratkaisu, kuten Github, Gitlab, BitBucket tai Azure DevOps toimii yksinkertaisimmillaan hajautettuna tietovarastona ohjelmakoodille. Tämä on huomattava parannus resilienssiin, koska hajautetussa tietovarastossa tallennus on redundanttia: yhden fyysisen komponentin hajoaminen ei aiheuta tiedonmenetystä [52].

Hajautettuna tietovarastona toimimisen lisäksi versionhallinta vastaa myös ohjelmakoodin virhealttiuteen. Oikein käytettynä versionhallinta helpottaa huomattavasti virheellisen ohjelmakoodin löytämisessä, koska virhe voidaan eristää pienempään päivitykseen. Virheistä palautuminen on myös helpompaa, koska ohjelmakoodi voidaan palauttaa tarvittaessa aikaisempaan, toimivaan versioon. Ominaisuuksia voidaan myös toteuttaa helposti uusille repositorion haaroille, jolloin toiminnallisuutta voidaan kehittää, testata ja laatuvarmistaa eristetysti. Näin voidaan helpottaa yhtäaikaista työskentelyä, parantaa ohjelmakoodin laatua sekä lisätä resilienssiä virheille. [21]

4.1.1 Toteutus

Projektiin valittiin versionhallinnaksi Git. Valinnan pääperusteena oli konsultin vahva kokemus Gitin käytöstä. Lisäksi Gitille on suosionsa ansiosta tarjolla huomattavasti muita vaihtoehtoja enemmän tukea ja apuohjelmistoja, -työkaluja sekä -järjestelmiä. Eräs apujärjestelmä on etärepositorio, joksi valittiin Gitlab. Gitlab on Githubin ohella suosituin Git-etärepositorio, ja myös sen valinta perustui suureksi osaksi konsultin aiempaan kokemukseen. Gitlab myös tarjoaa kilpailukykyistä ominaisuuskokoelmaa ilmaiseksi pienyrityksille, kuten projektin asiakkaalle.

Asiakkaan ohjelmistokehittäjällä ei ollut aiempaa kokemusta minkään versionhallinnan käytöstä. Gitin ja Gitlabin käyttöönotto tapahtui konsultin tuella. Käyttöönotto aloitettiin

luomalla koodipohjasta mahdollisimman puhdas versio ilman asiakaskohtaisia mukautuksia tai keskeneräisiä ominaisuuksia. Ohjelmakoodista luotiin Git-repositorio paikallisesti. Gitlabiin luotiin etärepositorio, joka yhdistettiin paikalliseen repositorioon ja ohjelmakoodit siirrettiin.

4.1.2 Koulutus

Pelkkä työkalu ei takaa versionhallinnan, kuten minkään muunkaan tietojärjestelmän, hyödyllisyyttä. Uuden järjestelmän käyttöönotto vaatii myös koulutusta, ei vain teknisestä käytöstä, vaan myös tehokkaaksi todetuista työskentelytavoista. [13] Käsiteltävän projektin tapauksessa koulutus aloitettiin täysin perusasioista.

Konsultilla, eli tutkijalla itsellään, on aikaisempaa kokemusta gitin käytön kouluttamisesta toiselle ohjelmistokehittäjälle. Konsultilla ei kuitenkaan ole tietämystä tai osaamista opetusmenetelmistä, joten niiden valinta ja soveltaminen perustui pääasiassa omaan kokemukseen. Tutkimuksen näkökulmasta tarkastellessa tässä havaitaan selkeä kehityskohta - projektin alussa olisi ollut hyödyllistä perehdyttää konsulttia kouluttamiseen. Toinen vaihtoehto olisi ollut hyödyntää konsulttia kattavammalla kokemuksella ja tietämyksellä kouluttamisesta ja koulutusmenetelmistä.

Suurin osa koulutuksesta tapahtui koulutuspalaverien muodossa. Nämä palaverit toteutettiin etätapaamisina Google Meet -alustalla. Konsultti valmisteli kokousten sisällön etukäteen. Kokouksen eteneminen oli kuitenkin joustavaa, ja sitä säädeltiin kehittäjän tarpeiden mukaan. Konsultti lopetti koulutuskokoukset selvittämällä kehittäjälle epäselväksi jääneet asiat, mitä hyödynnettiin seuraavan koulutuspalaverin valmistelussa. Kokouksista ei tuotettu perinteisiä kokousmuistioita, vaan tietoartifakteina tuotettiin yksityiskohtaisia ohjesivuja. Nämä ovat kokouksen jälkeen sekä konsultin että kehittäjän saatavilla.

Koulutetut aihealueet

Versionhallinnan kouluttaminen aloitettiin perustavanlaatuiselta tasolta. Ensimmäisessä koulutuspalaverissa lähdettiin liikkeelle siitä, mikä on versionhallinta, ja mitä sillä voidaan saada aikaan. Konsultti myös kartoitti kehittäjän aiempaa kokemusta, ja totesi että sitä ei ole.

Gitin rakenne voi olla vaikea ymmärtää kehittäjälle, joka ei tunne sitä ennestään. Väärin käytettynä Git voi myös olla sekä vaarallinen että sekava. Gitin tapa erotella committeja eri haaroille on usein vaikea hahmottaa, ja haarat saattavat sekoittua oppijan mielessä perinteisen unix-kansiorakenteen kansioihin. [27] Toinen yleinen haaste on paikallisen repositorion ja etärepositorion yhteys. Etärepositorioon liitetyssä Gitissä repositoriosta on kaksi versiota, paikallinen ja etärepositorion versio. Näiden tilat voidaan päivittää vastaamaan toisiaan komennoilla

```
git fetch
```

ja

```
git push
```

Usein fetch-komennosta käytetään aliasta,

```
git pull
```

joka suorittaa peräkkäin komennot

```
git fetch
```

ja

```
git merge
```

Näitä yleisesti havaittuja oppimisen haasteita tavattiin myös tutkittavassa projektissa. Lisäksi komentorivikäyttöliittymät olivat uusi aihe kehittäjälle. Git on oletuksena suunniteltu komentoriviltä käytettäväksi, mutta tarjolla on kolmannen osapuolen työkaluja graafisilla käyttöliittymillä. Kehittäjä otti käyttöön TortoiseGit-työkalun, mikä aiheutti koulutuksessa haasteita komentorivikäyttöön tottuneelle konsultille. Gitin komentoriviltä käytettäväksi suunniteltu toiminnallisuus kuitenkin sai kehittäjän siirtymään komentorivikäyttöliittymään myöhemmässä vaiheessa projektia, hänen käytettyään noin kolme kuukautta pääasiallisesti TortoiseGitin graafista käyttöliittymää.

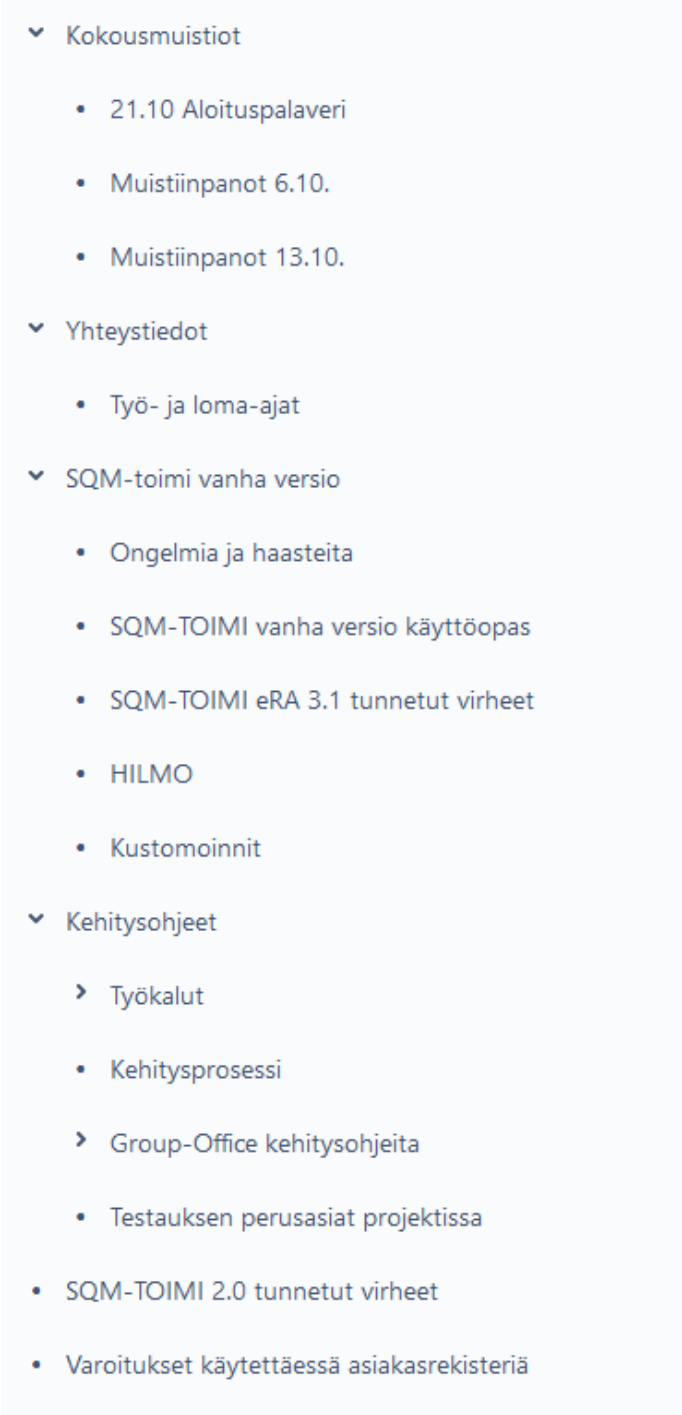
4.2 Dokumentaatio

Dokumentaation tuonti projektiin sisälsi sekä työskentelytapojen koulutusta että työkalujen käyttöönottoa. Projektilla ei ollut sisäistä dokumentaatiota, kuten luvussa 3 mainitaan. Sisäinen dokumentaatio toimisi projektissa kehityksen tukena.

4.2.1 Projektiwiki

Dokumentaatiotyökaluksi valikoitui verkkopohjainen wiki-alusta Confluence. Valinnan perusteena oli Gitlabin tapaan ilmaisen tason kattavat ominaisuudet sekä konsultin aiempi kokemus. Myöskään dokumentaatiosta kehittäjällä ei ollut aiempaa kokemusta, mutta motivaatio sen käyttöönottoon voitiin huomata - kehittäjä ja asiakasyrityksen johto tiedostivat dokumentaation hyödyllisyyden.

Dokumentaation hyödyntäminen aloitettiin projektissa Confluence-wikin käyttöönotolla. Samoin kuin versionhallinnan tapauksessa, käyttöönoton teki kehittäjä, jota konsultti ohjeisti etäyhteydellä reaaliajassa. Confluence wikiin suunniteltiin alustava rakenne, joka kehittyi projektin edetessä. Wikisivuston rakenne kirjoitushetkellä on esitetty kuvassa 4.1.

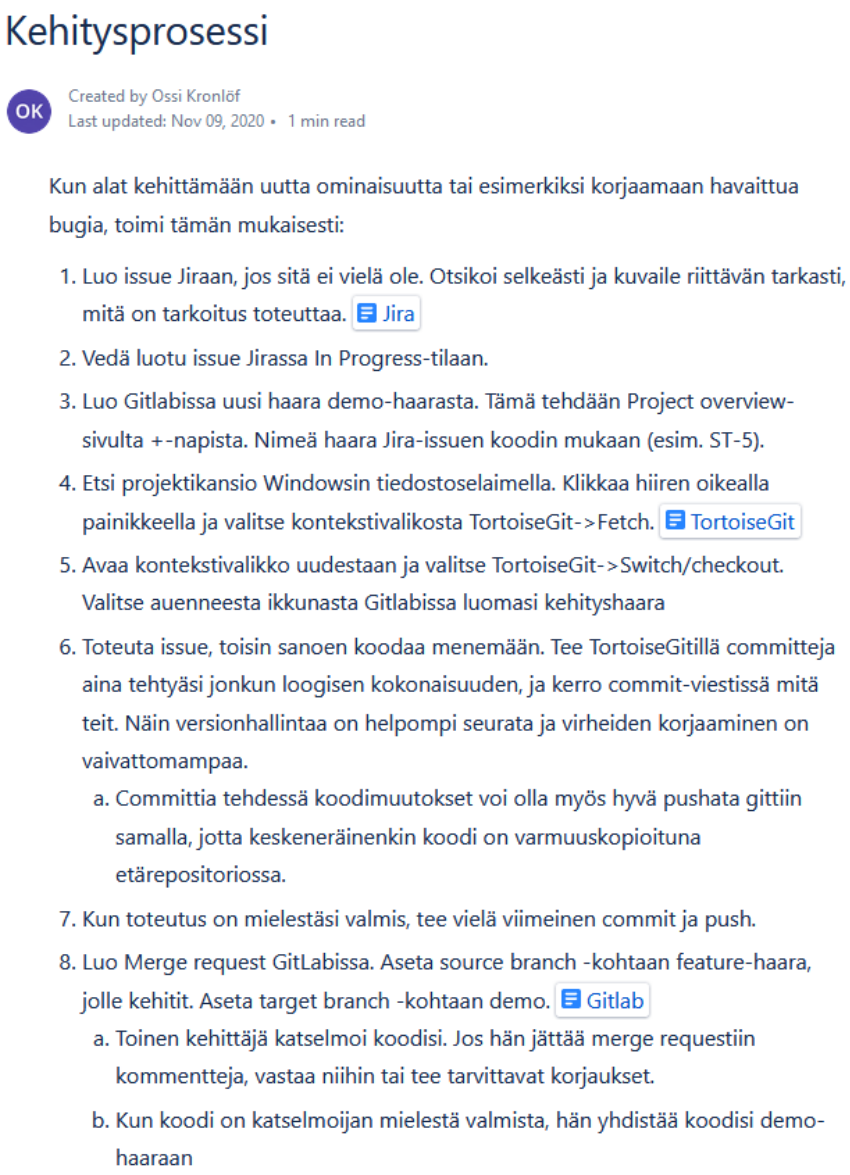


Kuva 4.1. *Projektin wikin rakenne*

Confluence wikin peruskäyttö koulutettiin kehittäjälle. Sen graafinen käyttöliittymä on suunnattu vähemmän tietoteknistä osaamista omaaville käyttäjille, joten oppiminen sujui enimmäkseen intuitiivisesti - konsultin lisäohjausta ei juurikaan tarvittu wikin teknisen käytön omaksumisessa. Kehittäjällä oli joitain haasteita oli tekstin formatoinnissa, mutta hän löysi niihin ratkaisuja myös ulkopuolisista tietoresursseista, kuten Confluencen dokumentaatiosta.

4.2.2 Dokumentaatiokäytännöt

Dokumentaatiokäytäntöjen implementointi vaati enemmän ohjausta ja koulutusta konsultilta. Eniten projektiwikiä hyödynnetään erilaisiin ohjesivuihin, jotka toimivat myös tiedonsiirron tukena konsultin ja kehittäjän välillä. Kuvan 4.1 osio Kehitysohjeet->Työkalut sisältää käyttöön otettujen kehitystyökalujen ohjesivuja. Nämä ohjeet on tuotettu koulutuspalavereiden aikana ja jälkikäteen niiden tulosten perusteella. Kuvassa 4.2 kuvatulla sivulla esitetään askeleittain, miten yksittäiseen kehitystehtävään liittyvä tehtävienhallintaprosessi tehdään. Ohjesivut toimivat koulutuksen tukena, ja niitä tuottavat sekä kehittäjä että konsultti.



Kehitysprosessi

OK Created by Ossi Kronlöf
Last updated: Nov 09, 2020 • 1 min read

Kun alat kehittämään uutta ominaisuutta tai esimerkiksi korjaamaan havaittua bugia, toimi tämän mukaisesti:

1. Luo issue Jiraan, jos sitä ei vielä ole. Otsikoi selkeästi ja kuvaile riittävän tarkasti, mitä on tarkoitus toteuttaa. Jira
2. Vedä luotu issue Jirassa In Progress-tilaan.
3. Luo Gitlabissa uusi haara demo-haarasta. Tämä tehdään Project overview-sivulta +-napista. Nimeä haara Jira-issuen koodin mukaan (esim. ST-5).
4. Etsi projektikansio Windowsin tiedostoselaimella. Klikkaa hiiren oikealla painikkeella ja valitse kontekstivalikosta TortoiseGit->Fetch. TortoiseGit
5. Avaa kontekstivalikko uudestaan ja valitse TortoiseGit->Switch/checkout. Valitse auenneesta ikkunasta Gitlabissa luomasi kehityshaara
6. Toteuta issue, toisin sanoen koodaa menemään. Tee TortoiseGitillä committeja aina tehtyäsi jonkun loogisen kokonaisuuden, ja kerro commit-viestissä mitä teit. Näin versionhallintaa on helpompi seurata ja virheiden korjaaminen on vaivattomampaa.
   a. Committia tehdessä koodimuutokset voi olla myös hyvä pushata gittiin samalla, jotta keskeneräinenkin koodi on varmuuskopioituna etärepositoriossa.
7. Kun toteutus on mielestäsi valmis, tee vielä viimeinen commit ja push.
8. Luo Merge request GitLabissa. Aseta source branch -kohtaan feature-haara, jolle kehitit. Aseta target branch -kohtaan demo. Gitlab
   a. Toinen kehittäjä katselmoi koodisi. Jos hän jättää merge requestiin kommentteja, vastaa niihin tai tee tarvittavat korjaukset.
   b. Kun koodi on katselmoijan mielestä valmista, hän yhdistää koodisi demo-haaraan

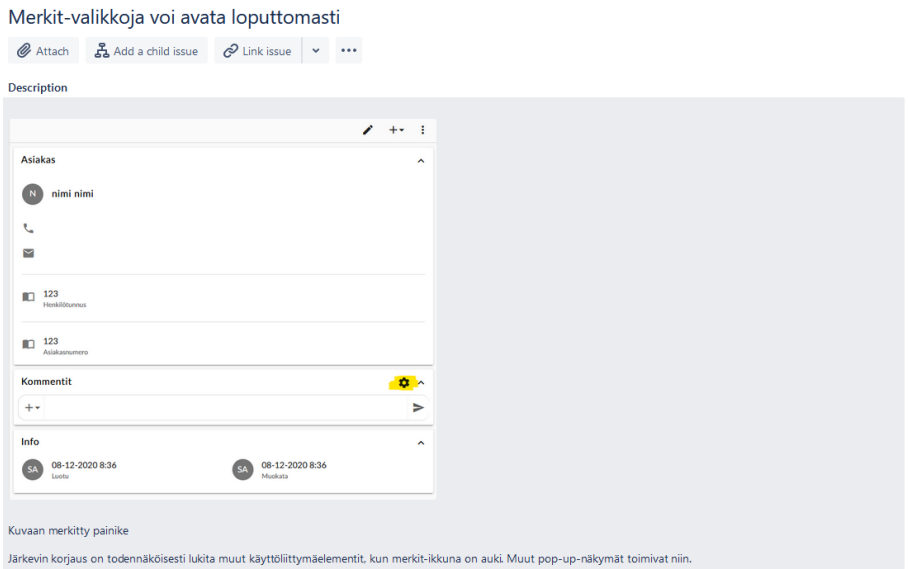
Kuva 4.2. *Ohjesivu kehitysprosessin kulusta projektin wikistä*

4.3 Jira

Projektissa otettiin käyttöön Jira -verkkoalusta tehtävienhallinnan työkaluksi. Projektissa ei ollut minkäänlaista tehtävänhallintaa, joten käyttöönotto ei vaatinut tietojen siirtämistä alustalta toiselle. Tämä helpotti käyttöönottoa, mutta toisaalta aiheutti lisäkoulutustarvetta tehtävienhallinnan osalta. Myös Jiran käyttöönotto tehtiin samaan tapaan etäpalaverissa kehittäjän toimesta konsultin ohjauksessa. Jira valittiin kehittäjän kokemuksen, laajan käyttäjäkunnan ja hyvien ilmaisominaisuuksien perusteella. Lisäksi Jira tarjoaa yhteentoimivuutta Confluence wikin kanssa, sillä molemmat ovat Atlassianin kehittämiä ohjelmistoja.

4.3.1 Kehitystehtävien pilkkominen Jira-tehtäviksi

Tehtävienhallinnan käyttöönotto vaati kehitystehtävien kuvailun Jira-tehtäviksi. Tämä on työtehtävien jäsentelyn kannalta hyvä asia. Tehtävien avulla kehityksen tavoitteita voidaan jakaa pienemmiksi kokonaisuuksiksi, ja ne mahdollistavat tarkemman kehityksen ohjaamisen ja seurannan. [46] Jira-tehtävien sisältö on vapaasti määritettävissä, ja usein sen tukena hyödynnetään jotakin ketterää kehityskehystä, kuten Scrum tai Kanban. Tyypillinen projektin kehitystehtävä on esitetty kuvassa 4.3. Kyseessä on bugikorjaus-tehtävä, ja hyvän käytännön mukaisesti tehtävään on liitetty kuvakaappaus esittämään missä virhe havaittiin.



Kuva 4.3. *Bugikorjaus-tehtävä projektin Jirasta*

4.3.2 Tehtävätaulun ylläpito

Myös tehtävätaulun rakenne on muokattavissa. Normaalisti tehtävätaulu koostuu nimetyistä sarakkeista, joiden alle listataan tehtäväkortteja. Sarakkeet kuvaavat sisältämiensä

tehtävien tilaa. Tehtävätaulun käyttäjäkokemus on myös korkeatasoinen, mikä näkyy korkeana asiakastyytyväisyytenä [11]. Tehtävätaulun käyttöönotto ei täten vaatinut suurta määrää koulutusta. Kehittäjän kanssa käytiin läpi uusien tehtävien luonti ja niiden siirtely tehtävätaululla. Haasteita ei juurikaan havaittu teknisessä käytössä.

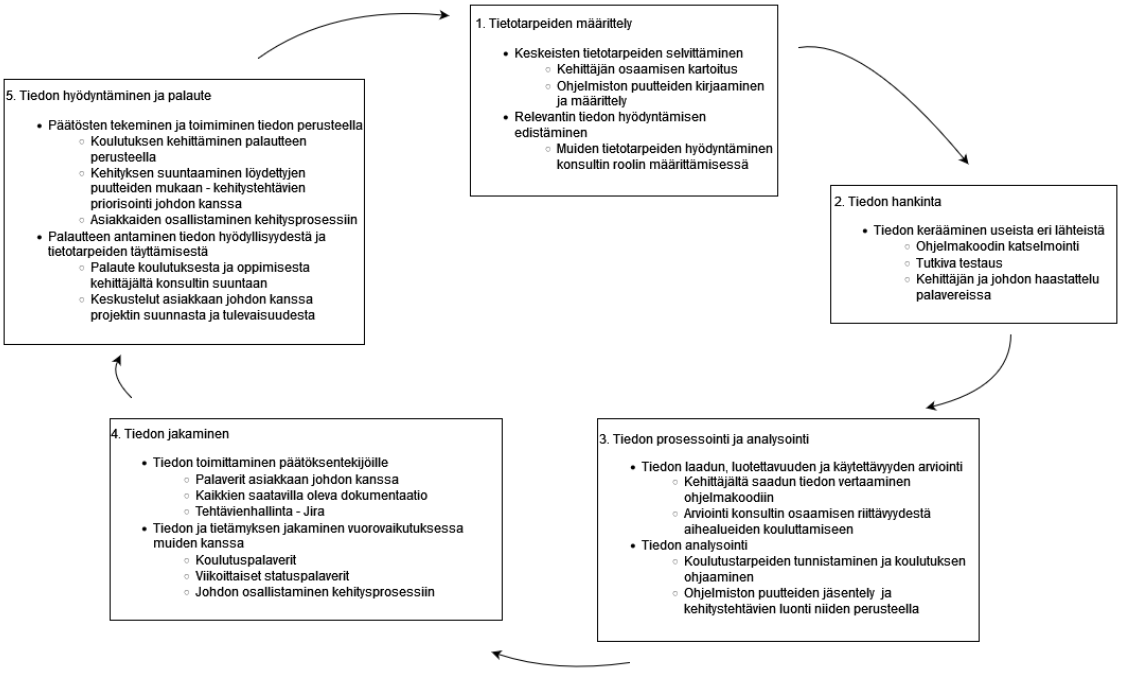
4.4 Mattermost

Mattermost otettiin projektissa käyttöön pääasiassa pikaviestimen rooliin. Mattermost mahdollistaa kommunikaation tarkemman jäsentelyn ja nopeammat vasteajat verrattuna sähköposteihin. Kommunikaatiokynnys on Mattermostin kaltaisessa pikaviestimessä matalampi kuin sähköposteissa tai (etä)tapaamisen järjestämisessä. Näin Mattermost fasilitoi tiedonsiirtoa ja lisähyötynä tieto muuttuu siirtyessään eksplisiittiseksi. Keskustelulangat ja -kanavat sekä tehokas hakutyökalu tukevat tiedon tarkastelua keskustelun jälkeen. [37] Perustelut Mattermostin valinnalle olivat samat kuin muillekin Atlassianin verkkoalustoille: hyvät ilmaisominaisuudet ja konsultin aiempi kokemus.

5. TIEDON JA OSAAMISEN HALLINTA JA KEHITYS

5.1 Tietotarpeiden määrittäminen, tiedonkeruu ja analyysi

Tiedonhallinnan kehitysprojektin ensimmäinen vaihe on tietotarpeiden määrittäminen [50]. Tietotarpeiden määrittäminen on yleensä tietojohtajan (information manager) vastuulla [18]. Tutkittavassa projektissa tässä roolissa toimi konsultti itse. Tietotarpeiden määritys on osa tiedonhallinnan prosessia. Seuraavana askeleena toimii tiedonkeruu, josta edetään analyysiin. Tiedonhallinnan prosessi tutkittavan projektin kontekstissa on esitetty kuvassa 5.1.



Kuva 5.1. *Tiedonhallinnan prosessi tutkittavassa projektissa. Mukautettu lähteestä [50]*

5.1.1 Kehittäjän osaamisen kartoitus

Tärkeä osa tietotarpeiden määrittämistä oli asiakkaan kehittäjän osaamisen kartoitus. Tietämystä kehittäjän osaamisesta voidaan käyttää analyysin tukena koulutustarpeiden selvittämisessä. Analyysin tulokset ohjaavat järjestettävän koulutuksen määrää ja sisältöä.

Kehittäjän osaaminen vaikuttaa myös konsultin rooliin. Havaitut koulutustarpeet vaikuttavat konsultin rooliin epäsuorasti asettamalla vaatimuksia konsultin osaamiselle ja suuntaamalla tiedonhankintaa. Lisäksi puutteet kehittäjän osaamisessa ohjaavat konsultin teknistä työtä suorasti. Osa-alueet, joissa konsultin osaaminen on vahvaa ja kehittäjän osaaminen heikkoa, muodostavat tarpeen tekniselle konsultoinnille ja toteuttavalle työlle konsultin osalta.

Kehittäjän osaamista kartoitettiin täysin kvalitatiivisesti. Konsultti keskusteli kehittäjän kanssa erityisesti projektin alkuvaiheen palavereissa kehittäjän osaamispohjasta. Keskustelua käytiin paljon limittäin asiakkaan konsultointitarpeen tarkentamisen ohessa. Kehitystarvetta havaittiin teknisessä osaamisessa, kuten asiakas odottikin. Kuitenkin suurempana haasteena havaittiin tietämyksen puute ohjelmistotuotannon työkaluista ja käytännöistä, kuten ohjelmistokehitysmalleista, tehtävienhallinnasta, versionhallinnasta sekä dokumentoinnista.

5.1.2 Ohjelmiston puutteiden määritys ja jäsentely

Tietotarpeiden määrittäminen oli tärkeää myös teknisen toteutuksen osalta. Koska ohjelmistokehitystä ei ollut dokumentoitu, eikä siitä ollut tuotettu muitakaan tietoartifakteja, ohjelmiston puutteet olivat vain hiljaisena tietona kehittäjällä itsellään. Hiljainen tieto ei ole organisaatiolle yhtä käyttökelpoista tai arvokasta kuin rakennepääomaan luettava dokumentoitu tieto [50, 32]. Rajallisen teknisen osaamisen myötä oli todennäköistä, että ohjelmistossa on myös tiedostamattomia puutteita.

Ohjelmiston puutteita tarkasteltiin useampaa kautta. Hiljaista tietoa haasteista ja virheistä dokumentoitiin projektin wikiin ja muutettiin näin eksplisiittiseksi. Tämä tapahtui kehittäjän toimesta konsultin ohjeilla. Dokumentoiduista puutteista määritettiin yhteistyössä asiakkaan johdon, konsultin ja kehittäjän kanssa tavoitteita kehitykselle. Näitä tavotteita pilkottiin kehittäjän ja konsultin kesken kehitystehtäviksi, joista luotiin yksittäisiä kehitystehtäviä Jiraan. Kehitystehtävien jäsentely ja dokumentointi toimi sekä hiljaisen tiedon muuttamisessa eksplisiittiseksi, että kehityksen strukturointina ylimääräisen työn vähentämiseksi ja työtehtävien selkeyttämiseksi. Alan kirjallisuus tukee toimintatavan hyötyjä sekä kehittäjän työmotivaation ja tehokkuuden että ohjelmakoodin laadun parantamisessa [29, 14, 43].

Tiedostamattomia puutteita etsittiin testauksella ja katselmoinnilla. Konsultti teki sekä itsenäisesti että yhteistyössä asiakkaan kehittäjän kanssa tutkivaa testausta ohjelmistolle. Testausta tehtiin sekä käsin, että rajallisesti Robot Framework -työkalulla. Tutkivassa testauksessa löytyneitä virheitä dokumentoitiin Jira-tehtävienhallintaan "bug-tyyppisinä tehtävinä.

Tutkivan testauksen lisäksi ohjelmistolle tehtiin koodikatselmointia. Asiakkaan kehittäjä

ohjasi konsulttia havaitsemiinsa ongelmakohtiin, ja parannusehdotuksia dokumentoitiin ohjelmakoodikommentteina, ja uusien ominaisuuksien tapauksessa Gitlabin katselmointityökalua hyödyntäen. Palaverien yhteydessä konsultti myös pyysi kehittäjää esittelemään ja selittämään ohjelmakoodin ongelmakohtia.

5.1.3 Konsultin rooli projektissa

Konsultin rooli projektissa oli yksi tärkeimpiä ja kiireellisimpiä tietotarpeita projektin alussa. Konsultointia hakiessaan asiakas tiedosti tarpeen tekniselle konsultoinnille: ohjelmoinnissa kohdattavien haasteiden ratkaisemiselle ja kehittäjän kouluttamiselle projektissa käytettävien ohjelmointikielten osalta. Kehittäjän osaamisen kartoitus auttoi määrittämään myös konsultin roolia paremmin. Tukea tarvitaan myös ohjelmistotuotannon käytäntöihin sekä koulutuksen että käyttöönoton muodossa.

Kehittäjän osaamisen kartoituksen ja ohjelmiston laadullinen tarkastelun analyysit loivat pohjan tietotarpeiden määrittämiselle konsultin roolin osalta. Alkukäsitys konsultin vastuusta projektissa puhtaasti teknisen toteutuksen tukena laajeni osaamisen kehittämiseen ja tiedonhallintaan - kokonaisvaltaiseen tiedon ja tiedolla johtamisen rooliin. Konsultin tehtäviin sisältyi myös teknistä toteutusta, ohjelmakoodin laadun kehittämistä sekä testausta. Suuri osa työtunneista kuitenkin ohjautui kehitysprosessin parantamiseen ja koordinointiin sekä asiakkaan kehittäjän ohjaukseen ja koulutukseen.

5.2 Osaamisen kehittäminen

Osaamisen kehittäminen oli merkittävä osa konsultin vastuita projektissa. Osaamisen kehittäminen projektin mikrotiimin tapauksessa rajautui käytännössä yhteen kehittäjään. Kehittäjän osaamisen kartoittaminen toimi tukena osaamisen kehittämisen toimenpiteiden valinnassa ja koulutuksen suuntaamisessa.

5.2.1 Koulutuspalaverit

Tärkeänä työkaluna osaamisen kehittämiselle toimivat koulutuspalaverit. Kehityskäytäntöjä ja -työkaluja esiteltiin kehittäjälle asteittain. Konsultti antoi kehittäjälle vastuun uusien alustojen, kuten Jiran, Confluencen, Gitlabin ja Mattermostin käyttöönotosta tukeakseen suorittamalla oppimista. Ohjelmistokehityksen käytäntöjä tuotiin mukaan uusien työkalujen lomassa, ja tiettyyn työkaluun liittyvät käytännöt pyrittiin esittelemään työkalujen yhteydessä. Tämän tarkoituksena oli tuoda esille työkalujen hyödyt, ja toiseen suuntaan esittää miten kehittäjälle uusia kehityskäytäntöjä voidaan konkreettisesti toteuttaa ja hyödyntää.

Esimerkkinä voidaan käyttää Jira-tehtävienhallintaa. Osana sen käyttöönottoa kehittäjäl-

le koulutettiin ketterää kehitystapaa. Kehittäjä ohjattiin luomaan Jiraan kanban-taulu. Samassa palaverissa koulutettiin mikä kanban on, mitä hyötyjä siitä voi saada ja miten Jiran kanban-taulua käytetään työtavan tukena. Lähestymistapa pyrkii luomaan kehittäjälle insentiivin muutokseen, tässä tapauksessa uuden kehitysmallin käyttöönottoon. Niin kutsutun vetävän (pull) muutosten fasilitoinnin on todettu kannustavan muutosten ylläpitoa erityisesti pitkällä tähtäimellä paremmin kuin työntävän (push) fasilitoinnin, jossa uusia työtapoja "pakotetaan"ottamaan käyttöön [24]. Projektin tapauksessa vetovaikutus pyrittiin tuomaan kehitystavan käyttöönottoon edellä kuvatusti selittämällä alustan käyttöönoton yhteydessä kehityskäytännön hyödyistä ja yhdistämällä työkalu vahvasti prosessiin.

Oppimisen tukena hyödynnettiin kertausta ja vastaanottavuutta esiin tuoduille vaikeuksille. Koulutuspalavereissa käytiin läpi yhteenveto edeltävän palaverin aiheesta. Samoja konsepteja käsiteltiin useammissa palavereissa. Avointa keskusteluilmapiiriä kannustettiin informaalilla kommunikaatiolla ja pehmeällä suhtautumisella haasteisiin ja virheisiin. Tämä kannusti kehittäjää tuomaan esille kohdatut ongelmat aikaisemmin koulutetuissa aiheissa, jolloin aiheita voitiin tarvittaessa kerrata uudestaan tai kouluttaa syvemmin. Kuvattu toimintatapa tukee oppimista selventämällä opittua, vahvistamalla muistijälkeä opituista aiheista ja toimimalla tukena mentaalimallien luomisessa [12].

5.2.2 Koodin katselmointi

Koodin katselmointi toimii laadunvarmistuksen työkaluna. Se ei kuitenkaan jäänyt toimintatavan ainoaksi rooliksi. Koodikatselmointia hyödynnettiin projektissa myös osaamisen kehittämisen tukena. Katselmointi antoi konsultille mahdollisuuden syventyä kehittäjän ohjelmointityyliin ja teknisen osaamisen puutteisiin. Erityisesti gitlabin katselmointityökalu tuki palautteenantoa kehittäjän koodista. Katselmointityökalu mahdollistaa helpon vertailun vanhan ja uuden ohjelmakoodiversion välillä sekä kommenttien jättämisen tiettyyn muutokseen kohdistettuna. Koodikatselmoinnin on todettu parantavan ohjelmiston laatua oikein käytettynä sekä fasilitoivan kehittäjien jatkuvaa oppimista [43].

5.2.3 Pariohjelmointi

Haastavissa teknisissä aiheissa koodin katselmointi ei riittänyt ainoaksi työkaluksi. Pariohjelmointia hyödynnettiin vaikeimpien haasteiden ratkaisussa. Useimmin asiakkaan kehittäjä toimi pariohjelmoinnissa ajurina (driver) ja konsultti ohjaajana (navigator). Pariohjelmoinnissa toteutettiin ominaisuuksia etäyhteyden välityksellä yhteistyönä kehittäjän ja konsultin välillä. Pariohjelmointi toimi myös kahdensuuntaisen tiedonsiirron tukena tapauksissa, joissa konsultti tarvitsi käytetystä teknologiastackista enemmän kokemusta omaavan kehittäjän tukea ohjelmakoodin ymmärtämiseen. Näin parikoodaus täytti tarkoitustaan toimimalla sekä tietämyksen kehittäjänä että tasoittajana, tuottaen uutta tietä-

mystä konsultin ohjelmistotuotannon osaamisen ja kehittäjän Group-Office -kokemuksen pohjalta, sekä tuoden kehittäjän tietämystä ohjelmistotuotannon menetelmistä ja parhaista käytännöistä riittävälle perustasolle yhteistyötä varten. [10]

6. JOHTOPÄÄTÖKSET JA KESKUSTELU

6.1 Työkaluvalinnat

Projektin työkaluvalinnat tehtiin suureksi osaksi kehittäjän aiemman kokemuksen perusteella. Konsultin projektiin käytettävän aikaikkunan rajallisuus estää työkaluvalintojen pitkäaikaisvaikutusten empiirisen tarkastelun. Joidenkin työkalujen osalta valintojen odotettua toimivuutta voidaan kuitenkin tarkastella teoriapohjalta. Muun muassa Git on hyvin laajassa käytössä versionhallintana kilpailijoihinsa nähden - sen markkinaosuus on joidenkin lähteiden mukaan jopa 70 % [38]. Yleisesti versionhallinnan käytön hyödyt on todettu alan kirjallisuudessa, ja niitä käsiteltiin luvussa 4 [27]. Nimenomaan gitin hyödyt on myös todettu tutkimuksessa [21].

Sen sijaan Atlassianin SaaS-ratkaisujen, eli Jiran, Confluence Wikin ja Mattermostin valinta tehtiin kevyin perustein. Vaihtoehtoisiin ratkaisuihin tutustuminen ei ollut syvällistä, johtuen ajan ja kokemuksen puutteesta. Projektin aikana työkalujen todettiin empiirisesti toimivan hyvin sekä konsultin että kehittäjän toimesta, mutta teoreettista tukea on haastava löytää, ja näin työkalujen soveltuvuutta on haastava arvioida pitävästi.

Työkaluvalintojen merkitys on kuitenkin pieni suhteessa prosesseihin ja käytäntöihin niiden ympärillä. Suurimmat potentiaaliset edut voidaan saavuttaa projektin aikana tapahtuneella organisaation oppimisella, organisaatiokulttuurin muokkaamisella ja osaamisen kehittämisellä. Työkalujen tärkein rooli on käytäntöjen muutoksen tukena. Jira mahdollistaa ketterän työtavan hyödyntämisen, samoin kuin muutkin vastaavat työkalut, kuten Microsoft TFS [11]. Confluence mahdollistaa dokumentaatiokäytäntöjen luomisen ja näin hiljaisen tiedon muuttamisen eksplisiittiseksi, toiminnon joka olisi mahdollista saavuttaa muillakin wikisivustoilla. Mattermost madaltaa kommunikaatiokynnystä samalla tavoin kuin slack, ja fasilitoi näin tiedon siirtoa, jalostamista ja jakamista sekä hiljaisen tiedon muuttamista eksplisiittiseksi [37]. Kaiken tämän tukena ja pohjana on muutosjohtaminen - työskentelykäytäntöjen muutos ja jatkuva oppiminen pitää rakentaa osaksi organisaatiokulttuuria [24].

Ylläpito ja jatkuva kehitys

Muutosten pysyvyyttä ja jatkuvaa oppimista on haastavaa mitata rajatun aikaikkunan vuoksi. Muutoksenhallinnan näkökulmasta katsottuna jotkin projektin piirteet kasvattavat todennäköisyyttä muutoksen pysyvyyteen, toiset taas eivät. Muutosvastarinta oli olematonta, mikä viittaa muutosmyönteiseen organisaatiokulttuuriin. Kehittäjä oli vastaanottavainen uusien käsitteiden oppimiselle ja motivaatio muutoksiin oli suuri. Projektin alkua edeltävän työtavan aiheuttamia ongelmia oli havaittu ja niihin kaivattiin muutosta. Johdon osallistaminen ei kuitenkaan ollut niin kattavaa kuin olisi toivottu, ja täten kehityksellä on vaarana jäädä organisaatiohierarkiassa matalalle tasolle - pelkästään kehittäjän toimintaan. [24]

Mattermostin käyttö viestintään ei vaikuttanut toteutuneen organisaation sisällä. Viestintäalusta oli enimmäkseen käytössä vain kehittäjän ja konsultin välisessä kommunikaatiossa. Johtoa olisi ollut hyvä osallistaa lisää alustojen käyttöönottoon ja tutustuttaa heidän rooleihinsa eri työkalujen käyttäjinä ja osana kehitysprosessia.

6.2 Osaamisen kehityksessä tehdyt valinnat

Konsultilla ei ole koulutusta tai aiempaa kokemusta koulutuksesta tai koulutusmenetelmistä. Koulutusmenetelmien valinnalle ei siis ollut tieteellistä osaamisperustaa. Menetelmät valittiin tietojohtamisen opinnoista saadun tietämyksen ja aiemman kokemuksen perusteella. Koulutuksessa käytettiin hyväksi tekemällä oppimista, kertausta ja palautetta, jotka on todettu toimiviksi koulutusmenetelmiksi [12]. Myös ohjelmistotuotannon kirjallisuus tukee projektipohjaista oppimista, ketterää oppimista ja vertaisoppimista, joita kaikkia hyödynnettiin projektissa [42].

Osaamista tai oppimista ei mitattu tai testattu projektin aikana. Mittaus ja testaaminen olisivat mahdollistaneet hyödynnettyjen koulutusmenetelmien evaluoinnin projektin kontekstissa sekä paremman koulutuksen ohjaamisen eri aihealueisiin. Mittaus ja testaaminen jäivät tekemättä sekä puuttuvan tietämyksen että rajallisen ajan takia. Mittaus toimii myös muutoksenhallinnan tukena. [24]

Etätyöskentelyllä on vaikutus koulutukseen ja oppimiseen. Suomessa ja Intiassa toteutettu tutkimus osoittaa yliopisto-opiskelijoiden kokeneen opetuksen sekä lyhyellä että pitkällä tähtäimellä matalalaatuisemmaksi siirryttyään etäopiskeluun [51]. Projektin jäsenet kuitenkin olivat kokeneita etätyön tekijöitä, ja negatiivisia puolia pyrittiin minimoimaan myös laajalla digitaalisten vuorovaikutusalustojen hyödyntämisellä [37].

Pysyvyys

Muutosten pysyvyyttä ei voida arvioida empiirisesti rajatun aikaskaalan takia. Muutoksen pysyvyydelle on tärkeintä organisaatiokulttuuri. Muutoksen pysyvyyttä rajoittavat johdon vähäinen osallistaminen sekä mittaamisen ja jatkuvan kehityksen puute. Kuitenkin jo projektia edeltävä muutosmyönteinen organisaatiokulttuuri on positiivinen voima muutoksen ylläpidon kannalta. [24]

6.3 Prosessien kehittämisen rajaus

Resurssien rajallisuuden vuoksi prosessien ja työkalujen kehittämistä piti rajata. Tähän ei varattu merkittävästi resursseja projektin alussa. Käytännössä tämä tarkoitti, että projektia ei rajattu tarpeeksi pieneen laajuuteen. Kehitykseen pyrittiin tuomaan liian isoa määrää kehityskäytäntöjä ja työkaluja. Tämä voitiin havaita empiirisesti jo projektin aikana. Ensinnäkin ohjelmistoa pyrittiin kontainerisoimaan Dockerin avulla. Se toimi kehityksen tukena yhtenäistämällä ajoympäristöt kehittäjien välillä, mutta palvelinympäristön rajoitteiden takia sitä ei saatu ikinä tuotantokäyttöön. Asiakkaan kehittäjän koulutus aiheesta jäi myös vajavaiseksi.

Toinen liian kunnianhimoinen uudistus oli projektiin tuotavien testauskäytäntöjen ja -työkalujen laajuus. Suureksi osaksi Group-Officen toteutuksen takia käyttöliittymätestaus Robot Frameworkilla ei toteutunut käytännössä, vaan sillä tehtiin vain kaksi esittelyluontoista testiä. Sekä kontainerisointiin että testien luomiseen sekä testauksen kouluttamiseen käytettiin resursseja saamatta vastaavaa hyötyä kuin suuntaamalla ne muihin projektin osa-alueisiin. Myös CI/CD, eli jatkuva integraatio ja julkaisu jäi toteuttamatta ja kouluttamatta laajasti. Sen kuitenkin havaittiin olevan liian laaja kokonaisuus jo ajoissa, ja resurssihukka oli pienempää.

Teknologiastackin säilyttäminen ennallaan oli onnistuneesti tehty rajaus projektin alkuvaiheessa. Ohjelmiston siirtäminen kokonaan uudelle pohjalle ja kehittäjän kouluttaminen olisi ollut resurssien puitteissa käytännössä mahdotonta. Olisi kuitenkin ollut projektin tulevaisuuden kannalta parempi, jos resursseja olisi voitu kasvattaa tarpeeksi ohjelmiston toteuttamiseksi alusta alkaen nykyaikaisilla teknologioilla, kuten node.js ja react -stackilla. Niin sanotun legacyohjelmiston ylläpito kasvattaa teknistä velkaa, ja Group-Officen jo valmiiksi pienen käyttäjäkunnan pienentyessä tietoresurssien ja (vertais)tuen saatavuus heikentyy [9].

6.4 Konsultin osaamisen puutteet ja odottamaton rooli

Konsultti päätyi projektissa sekä johtamis- että koulutustehtäviin kehitystehtävien sijaan. Tämä on yleinen tilanne ohjelmistoalan konsulttiprojekteissa [8]. Tässä projektissa kon-

sultin osaamisalue tuki odottamatonta roolia tietojohtamisen opintojen ja oman kiinnostuksen myötä. Tietämys koulutusmenetelmistä oli puutteelista. Konsultilla oli projektin alussa osaamista web-kehityksestä, mutta ohjelmiston teknologiastack, Group-Office, ei ollut ennestään tuttu. Tämä toimii myös esimerkkinä legacy-koodin ylläpitämisen haasteista - on lähes mahdotonta löytää konsulttiapua ohjelmistoon täysin soveltuvalla teknologiaosaamisella.

Dokumentaatio oli myös osa-alue, johon konsultilla ei ollut koulutuspohjaa. Käyttöön otetut dokumentaatiokäytännöt perustuivat rajalliseen kokemukseen työelämästä. Tämä on ohjelmistokonsultin roolissa yleistä, ja liittyy myös odottamattomaan rooliin [8]. Dokumentaatioon liittyen erityisesti muistiinpanokäytännöt olivat puutteellisia. Kokousmuistioita ei tuotettu jokaisesta kokouksesta, ja niiden sisältö ei ollut yhtenäisesti strukturoitua. Muistiinpanojen tekeminen on kuitenkin hyödyllistä muistijäljen luomisessa, joten niiden kirjoittaminen oli silti hyödyllistä osaamisen kehittämisen näkökulmasta [22].

7. YHTEENVETO

Työssä tutkittiin SQM-Toimi -ohjelmiston uuden version kehitysprojektia tapaustutkimuksena. Tutkimus käsitteli konsultin näkökulmasta projektissa tehtyjä parannuksia kehityskäytäntöihin. Muutoksilla pyrittiin parantamaan ohjelmiston laatua sekä kehitystyön selkeyttä ja vähentämään ylimääräistä työtä, näin parantaen työn tehokkuutta. Koska pitkän aikavälin empiirinen tarkastelu ei ole mahdollista, muutoksia syvennytään tarkastelemaan verraten ohjelmistotuotannon ja tietojohtamisen kirjallisuuteen.

Projektissa tehtyjä työkaluvalintoja tarkasteltiin verraten ohjelmistotuotannon kirjallisuuteen ja vakiintuneisiin käytäntöihin. Työkaluvalinnat todettiin suurelta osin hyväksi, ja esimerkiksi Gitin käyttö versionhallintana on perusteltua sen vakiintuneen markkina-aseman ja vertailevassa tutkimuksessa todettujen hyötyjen myötä. Jotkin projektin työkaluvalinnoista on tehty kevyin perustein, ja pitkäaikaisen vaikutusten seurannan puuttuessa niitä ei voida juuri analysoida. Tutkimus kuitenkin näkee työkaluvalintoja tärkeämpänä prosessit ja käytännöt niiden ympärillä.

Tiedon ja osaamisen hallintaan liittyvät kehitystoimet projektissa ovat työkalujen ohessa tärkeä tarkastelun aihe. Projektissa hyödynnettiin tiedonhallinnan keinoja ohjelmiston puutteiden määrittämiseen ja jäsentelyyn sekä kehittäjän osaamisen kartoitukseen. Löydösten pohjalta rakennettiin kuva konsultin roolista projektissa. Tietojohtamisen kirjallisuuteen verrattaessa havaittiin, että konsultin ohjaus- ja koulutustehtäviin ohjautunut rooli odotetun teknisen konsultoinnin sijasta ei ole ohjelmistoalalla poikkeava tapaus, vaan usein havaittu ilmiö. Tulokset tukevat siis aiemman tutkimustiedon kanssa ohjelmistoalan konsulttiyritysten tarvetta kouluttaa konsulttejaan laaja-alaisemmin pelkän teknisen osaamisen sijaan. Tutkitussa projektissa erityisesti tietämykselle koulutusmenetelmistä havaittiin takautuvasti olleen enemmän tarvetta projektin alussa.

Lisäksi tutkimus tukee aiempaa tiedonhallinnan kirjallisuutta tiedonhallinnan prosessin tärkeydestä. Tarkasteltavassa projektissa tiedonhallinnan prosessille ei varattu tarpeeksi resursseja etupainotteisesti. Kattava tietotarpeiden määritys, tiedonkeruu ja tiedon analyysi projektin alussa olisivat auttaneet kohdentamaan resursseja projektin aikana ja toiminut kehityskohteiden arvottamisen ja rajauksen tukena.

Työkaluvalintojen sekä tiedon ja osaamisen hallinnan näkökulmien lisäksi tutkimus tarkastelee projektia muutosjohtamisen näkökulmasta. Parannusten ylläpito ja jatkokehitys

kyseenalaistetaan, koska projektin muutosjohtajana toiminut konsultti ei pysty tukemaan sitä omilla toimillaan pitkällä aikavälillä. Muutosjohtamisen kirjallisuuteen verrattaessa havaitaan, että jotkin piirteet projektissa ja projektiorganisaatiossa tukevat jatkuvaa organisaation oppimista. Erityisesti muutosmyönteinen organisaatiokulttuuri on merkittävä tekijä muutoksen pysyvyydessä. Tapa, jolla muutosta johdettiin projektissa oli myös tiedeperustainen, muun muassa vetävän muutosvoiman hyödyntämisessä työntävän sijaan. Kuitenkin jotkin tekijät, kuten johdon vähäinen osallistaminen luovat epäilystä muutoksen pysyvyydestä pitkällä aikavälillä.

LÄHTEET

[1] *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. eng. Project Management Institute, 2021.

[2] Ryan Angus, Matthew Barlow ja William S Schulze. ”Learning from Failed Innovation Experiments”. eng. Teoksessa: *Academy of Management Proceedings*. Vol. 2019. 1. 2019, s. 18909–.

[3] Atlassian. *Confluence Your remote-friendly team workspace*. URL: https://www.atlassian.com/software/confluence.

[4] Atlassian. *Jira: Issue amp; project tracking software*. 2022. URL: https://www.atlassian.com/software/jira.

[5] Atlassian. *Learn Kanban with jira software*. 2022. URL: https://www.atlassian.com/agile/tutorials/how-to-do-kanban-with-jira-software.

[6] Kent Beck et al. *Ketterän ohjelmistokehityksen julistus*. Toim. LasseTranslator Koskela. 2001. URL: https://agilemanifesto.org/iso/fi/manifesto.html.

[7] Emery Berger et al. ”On the Impact of Programming Languages on Code Quality: A Reproduction Study”. eng. *ACM transactions on programming languages and systems* 41.4 (2019), s. 1–24. ISSN: 0164-0925.

[8] Jason T. Bickle. ”Developing Remote Training Consultants as Leaders—Dialogic/Network Application of Path-Goal Leadership Theory in Leadership Development”. eng. *Performance improvement (International Society for Performance Improvement)* 56.9 (2017), s. 32–39. ISSN: 1090-8811.

[9] Chris Birchall. *Re-engineering legacy software*. eng. 1st edition. Shelter Island, New York: Manning, 2016. ISBN: 1-63835-332-8.

[10] Adrian Bolboaca. *Practical remote pair programming : best practices,tips, and techniques for collaborating productively with distributed development teams*. eng. London, England: Packt Publishing, Limited, 2021. ISBN: 1-80056-553-4.

[11] Marius-Constantin Brad et al. ”A Comparative Study of Agile Project Management Software Tools”. eng. *Economy informatics* 16.1 (2016), s. 27–38. ISSN: 1582-7941.

[12] Peter C. Brown. *Make it stick : the science of successful learning*. eng. Pilot project. eBook available to selected US libraries only. Cambridge, MA: Harvard University Press, 2014 - 2014. ISBN: 9780674419377.

[13] Nesrine Chtourou Ben Amar ja Randa Ben Romdhane. ”Organizational culture and information systems strategic alignment: Exploring the influence through an empirical study from Tunisia”. eng. *Journal of enterprise information management* 33.1 (2020), s. 95–119. ISSN: 1741-0398.

[14] David Conrad, Amit Ghosh ja Marc Isaacson. ”Employee motivation factors”. eng. *International journal of public leadership* 11.2 (2015), s. 92–106. ISSN: 2056-4929.

[15] N Cook ja D Yanow. ”Culture and Organizational Learning”. eng. *Journal of management inquiry* 20.4 (2011), s. 362–379. ISSN: 1056-4926.

[16] Jamie Lynn Cooke. *Agile: Real Results from IT Budgets*. eng. Ely: IT Governance Ltd, 2016. ISBN: 9781849287951.

[17] Anne Dohrenwend. ”Serving up the feedback sandwich: negative feedback is never easy to give, but sandwiching criticism between layers of praise makes it more palatable and more effective”. eng. *Family practice management* 9.10 (2002), s. 43–. ISSN: 1069-5648.

[18] Daniel G Dorner, G.E Gorman ja Philip J Calvert. *Information needs analysis: principles and practice in information organizations*. eng. London: Facet Publishing, 2017. ISBN: 185604484X.

[19] MARK EASTERBY-SMITH. *Handbook of organizational learning and knowledge management, second edition*. eng. Chichester, West Sussex, U.K, 2011.

[20] Marcos Fuentes, Hedley Smyth ja Andrew Davies. ”Co-creation of value outcomes: A client perspective on service provision in projects”. eng. *International journal of project management* 37.5 (2019), s. 696–715. ISSN: 0263-7863.

[21] Kenneth Geisshirt et al. *Git Version Control Cookbook: Leverage Version Control to Transform Your Development Workflow and Boost Productivity, 2nd Edition*. eng. Birmingham: Packt Publishing, Limited, 2018. ISBN: 9781789137545.

[22] Patrick Gourley. ”Back to basics: How reading the text and taking notes improves learning”. eng. *International review of economics education* 37 (2021), s. 100217–. ISSN: 1477-3880.

[23] J. Richard Hackman. ”Why teams don’t work”. eng. *Leader to leader* 1998.7 (1998), s. 24–31. ISSN: 1087-8149.

[24] John Hayes. ”The Theory and Practice of Change Management” (2014).

[25] Zhimin He, Hao Yang ja Min Xie. ”Statistical modeling and analysis of hard disk drives (HDDs) failure”. eng. Teoksessa: *2012 Digest APMRC*. IEEE, 2012, s. 1–2. ISBN: 1467347345.

[26] Kristina Heinonen et al. ”A customer-dominant logic of service”. eng. *International journal of service industry management* 21.4 (2010), s. 531–548. ISSN: 1757-5818.

[27] Ville Isomöttönen ja Michael Cochez. ”Challenges and Confusions in Learning Version Control with Git”. eng. Teoksessa: *Communications in computer and information science (Internet)*. Vol. 469. Communications in Computer and Information Science. Cham: Springer International Publishing, 2014, s. 178–193. ISBN: 3319132059.

[28] Michael D. Jacobson. ”Forget The Feedback Sandwich”. eng. *Family practice management* 25.5 (2018), s. 35–35. ISSN: 1069-5648.

[29] Petri Karkkola, Matti Kuittinen ja Taina Hintsa. ”Role clarity, role conflict, and vitality at work: The role of the basic needs”. eng. *Scandinavian journal of psychology* 60.5 (2019), s. 456–463. ISSN: 0036-5564.

[30] Chris Kraft. ”What Is SaaS and How Can It Help Build Our Profession?” eng. *The journal of government financial management* 67.2 (2018), s. 26–31. ISSN: 1533-1385.

[31] Center for Creative Leadership. *Feedback that works how to build and deliver your message.* eng. Second edition. The ideas into action series. Place of publication not identified: Center for Creative Leadership, 2019. ISBN: 1-60491-923-X.

[32] *Liiketoiminnan aineettomat menestystekijät mittaa, kehitä ja johda*. fin. Helsinki: Talentum, 2007. ISBN: 978-951-8968-64-4.

[33] Tania Luna ja LeeAnn Renninger. *The Leader Lab*. eng. Wiley, 2021. ISBN: 1119793319.

[34] John Mathieu et al. ”Team Effectiveness 1997-2007: A Review of Recent Advancements and a Glimpse Into the Future”. eng. *Journal of Management* 34.3 (2008), s. 410–476. ISSN: 0149-2063.

[35] *Mattermost: Open Source Collaboration for developers*. Toukokuu 2022. URL: https://mattermost.com/.

[36] Ralf Müller ja Rodney Turner. ”Leadership competency profiles of successful project managers”. eng. *International journal of project management* 28.5 (2010), s. 437–448. ISSN: 0263-7863.

[37] Sean A Newman ja Robert C Ford. ”Five Steps to Leading Your Team in the Virtual COVID-19 Workplace”. eng. *Organizational dynamics* 50.1 (2021). ISSN: 0090-2616.

[38] Openhub.net. *Compare repositories*. URL: https://www.openhub.net/repositories/compare.

[39] *Osaamisen johtaminen muutoksessa : ideoita ja kokemuksia toisen sukupolven knowledge managementin kehittelyyn*. fin. Helsinki, 2002.

[40] *Our story × supercell*. 2022. URL: https://supercell.com/en/our-story/#beginning.

[41] Luka Pavlic, Marjan Hericko ja Tina Beranic. ”An expert judgment in source code quality research domain—a comparative study between professionals and students”. eng. *Applied sciences* 10.20 (2020), s. 1–13. ISSN: 2076-3417.

[42] Katarina Pazur Anicic ja Zlatko Stapic. ”Teaching Methods in Software Engineering: Systematic Review”. eng. *IEEE software* (2022), s. 0-0. ISSN: 0740-7459.

[43] Ricardo Perez-Castillo ja Mario Piattini. ”Understanding the Impact of Development Efforts in Code Quality”. eng. *JOURNAL OF UNIVERSAL COMPUTER SCIENCE* 27.10 (2021), s. 1096–1127. ISSN: 0948-695X.

[44] Eduardo Salas, Dana E Sims ja C. Shawn Burke. ”Is there a “Big Five” in Teamwork?” eng. *Small group research* 36.5 (2005), s. 555–599. ISSN: 1046-4964.

[45] Scrum.org. *Professional scrum competency: Understanding and applying the scrum framework*. URL: https://www.scrum.org/professional-scrum-competencies/understanding-and-applying-scrum-framework.

[46] Pedro Serrador ja Jeffrey K Pinto. ”Does Agile work? — A quantitative analysis of agile project success”. eng. *International journal of project management* 33.5 (2015), s. 1040–1051. ISSN: 0263-7863.

[47] Roy K. Smollan ja Rachel L. Morrison. ”Office design and organizational change: The influence of communication and organizational culture”. eng. *Journal of organizational change management* 32.4 (2019), s. 426–440. ISSN: 0953-4814.

[48] Ahmad Tahmid et al. ”Code sniffer: a risk based smell detection framework to enhance code quality using static code analysis”. eng. *International Journal of Software Engineering, Technology and Applications* 2.1 (2017), s. 41–63. ISSN: 2053-2466.

[49] *The AMA handbook of project management*. eng. New York, 2014 - 2014.

[50] *Tietojohtaminen*. fin. Tampere, 2013.

[51] Rucha Tulaskar ja Markku Turunen. ”What students want Experiences, challenges, and engagement during Emergency Remote Learning amidst COVID-19 crisis”. eng (2022).

[52] Neal Weinberg. ”Pros and cons of cloud storage: Cloud storage offers many advantages over on-premises data storage. Scalability at the push of a button (up or down), accessibility from any device at any location, and pay-per-usage pricing are a few of the draws. But there are some potential drawbacks as well”. eng. *Network World (Online)* (2020). ISSN: 1944-7655.

[53] CW Von Bergen, Martin S Bressler ja Kitty Campbell. ”The sandwich feedback method: not very tasty”. *Journal of Behavioral Studies in business* 7.1 (2014).

[54] Hai-Yan Xu ja Ying Jiang. ”Determination of Code Quality Attribute for Complex User’s Comments”. eng. *Ruan jian xue bao* 32.7 (2021), s. 2183–2203. ISSN: 1000-9825.

[55] Mika Yrjölä et al. ”Leading Change A Customer Value Framework”. eng. Teoksessa: Tampere University Press, 2019.

[56] Nazatul Nurlisa Zolkifli, Amir Ngah ja Aziz Deraman. ”Version Control System: A Review”. eng. *Procedia Computer Science* 135 (2018), s. 408–415. ISSN: 1877-0509.