Nanna Halonen

Scrum-viitekehyksen soveltaminen kehitysprojektissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

2.3.2024

Tiivistelmä

Tekijä:	Nanna Halonen
Otsikko:	Scrum-viitekehyksen soveltaminen kehitysprojektissa
Sivumäärä:	21 sivua
Aika:	2.3.2024

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Tietojenkäsittely ja tietoliikenne
Ohjaajat:	Osaamisaluepäällikkö, Salonen Janne

Tässä opinnäytetyössä tarkastellaan Scrum-mallin käyttöönottoa eläkevakuutusyhtiö Veritaksen kehitysprojektissa. Prosessin aikana pyrittiin tunnistamaan ja arvioimaan kehitysprojektin vakiintuneita toimintatapoja, kehittämään uusia toimeksiantajan tarpeisiin sopivia käytäntöjä sekä tukemaan projektitiimin jäseniä mallin käyttöönotossa.

Työn teoreettisena viitekehyksenä käytetään ketterän ohjelmistokehityksen periaatteita, standardeja ja Scrum-opasta. SWEBOK (Software Engineering Body of Knowledge) ja IEEE (Institute of Electrical and Electronics Engineers) -standardit tarjoavat laajemman viitekehyksen ohjelmistotuotantoon ja projektinhallintaan, johon Scrum-malli voidaan integroida. Scrum-opas toimii käytännön oppaana, joka auttaa ymmärtämään Scrum-mallin perusteet, roolit ja vastuut sekä säännöt ja käytännöt, joita tiimin on noudatettava.

Scrum-mallin käyttöönotto tapahtui viiden kuukauden koejakson aikana syksyllä 2023. Koejakson avulla organisaatio arvioi mallin sopivuutta omiin tarpeisiinsa ja tunnisti mahdolliset haasteet sen käyttöönotossa. Työn tavoitteena on vastata keskeisiin kysymyksiin kehitysprosessin Scrum-tapahtumista, sovelluksen laadun varmistamisesta kehitysprosessin aikana sekä kehitetyn prosessin seurannasta ja arvioinnista.

Avainsanat: ketterä kehitys, scrum, sovelluskehitys

Abstract

Author:	Nanna Halonen
Title:	Adapting the Scrum framework for development project
Number of Pages:	21 pages
Date:	3 March 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Information and Communication Technologies (ICTs)
Supervisors:	Janne Salonen, Head of School

This engineering work examines the implementation of the Scrum framework in development project for the pension insurance company Veritas. During the process, the aim was to identify and evaluate established practices within the project, develop new practices tailored to the client's needs, and support project team members in adopting the model.

The theoretical framework of this work consists of principles of Agile Manifest, standards, and the Scrum Guide. SWEBOK (Software Engineering Body of Knowledge) and IEEE (Institute of Electrical and Electronics Engineers) standards provide a broader framework for software production and project management, into which the Scrum framework can be integrated. The Scrum Guide serves as a practical guide, aiding in understanding the fundamentals, roles, responsibilities, rules, and practices of the Scrum framework.

The implementation of the Scrum framework took place during a five-month trial period in the autumn of 2023. Through this trial period, the organization assessed the suitability of the framework for its own needs and identified potential challenges in its adoption. The objective of this work is to address key questions regarding the Scrum events in the development process, ensuring application quality throughout the development process, and monitoring and evaluating the developed process.

Keywords: agile, scrum, software development

Sisällys

1 Johdanto 1

1.1 Työn tarkoitus ja tavoitteet 2
1.2 Työn toimeksiantaja 3

2 Ohjelmistotuotanto 3

2.1 Ohjelmistuotannon elinkaari ja elinkaarimallit 6
2.2 Ketterä ohjelmistokehittäminen 10

3 Scrum-mallin käyttöönotto 11

3.1 Roolit ja vastuualueet 13
3.2 Sprintin tapahtumat 15

4 Laadunvalvonta 17

5 Jatkokehitysehdotukset 18

5.1 Tunnuslukujen seuranta 18
5.2 Määrittelyprosessin kehittäminen 19

6 Yhteenveto 19

Lähteet 21

1 Johdanto

Nopea teknologinen kehitys ja asiakastarpeiden jatkuva muuttuminen edellyttävät organisaatioilta joustavuutta ja tehokkuutta tuotekehityksessä. Erityisesti ohjelmistokehitysprojekteissa on tunnistettu tarve ketterille ja iteratiivisille työskentelymenetelmille, jotka mahdollistavat nopean reagoinnin muutoksiin. Tämä opinnäytetyö käsittelee Scrum-sovelluskehyksen käyttöönottoa eläkevakuutusyhtiö Veritaksen kehitysprojektissa.

Eläkeyhtiö Veritaksen verkkopalvelu on digitaalinen alusta, joka tarjoaa asiakkaille mahdollisuuden hoitaa eläkevakuutusasioitaan verkossa. Asiakkaat voivat käyttää verkkopalvelua tarkastellakseen ja hallinnoidakseen omaa eläkevakuutustietoaan, tehdä muutoksia vakuutuksiin, saada laskelmia ja arvioita sekä hoitaa muita eläkevakuutukseen liittyviä asioita helposti ja kätevästi verkossa. Verkkopalvelu on suunniteltu tarjoamaan asiakkaille helppokäyttöinen ja saumaton tapa hallita omia eläkeasioitaan. (Veritas 2023a; Veritas 2015a.)

Verkkopalvelua on uudistettu sen elinkaaren aikana useita kertoja. Sovelluksen nykyinen CGI:n toteuttama kehitysprojekti aloitettiin 2018 ja uusi verkkopalvelu julkaistiin vuonna 2019. Projektin merkittäviä vuosittaisia hankkeita ovat olleet:

- 2019 vakuutusasiakkaiden uusi verkkopalvelu
- 2020 sähköisten vakuutushakemusten uusinta ja tunnistuspalvelun käyttöönotto
- 2021 sähköiset eläkehakemusten uusinta
- 2022 eläkeasioiden hallinta
- 2023 verkkotyöeläkeote ja YEL-lakimuutokset
- 2024 asiakkaiden tuntemisen automatisointi.

Kehitysprojektin aikana ylläpidettävien palveluiden ja palveluihin liittyvien integraatioiden määrä on kasvanut merkittävästi, ja projektitiimin jäsenet ovat muutamia avainhenkilöitä lukuun ottamatta vaihtuneet. Selkeiden ohjeiden ja rutiinien puuttuminen on aiheuttanut haasteita muun muassa perehdytyksessä, tuotetun sovelluksen laadussa sekä projektitiimin jaksamisessa. Tarve toimintatapojen muutokselle on nostettu esiin sekä toimeksiantajan että toimittajan aloitteesta.

1.1 Työn tarkoitus ja tavoitteet

Työn tarkoituksena oli määrittää ja arvioida kehitysprojektissa vakiintuneita ohjelmistotuotantoprosessiin liittyviä toimintatapoja, kehittää ja sopia toimeksiantajan ja Scrum-työskentelymalliin soveltuvia uusia käytäntöjä sekä tukea projektitiimin jäseniä muutoksessa. Työn teoreettisena viitekehyksenä toimivat ketterän ohjelmistokehityksen julistus, SWEBOK (IEEE 2014; IEEE 2022), IEEE-standardit ja Scrum-opas (Schwaber & Sutherland 2020). Scrum-mallin käyttöönotto tapahtui 5 kuukauden koejakson aikana syksyllä 2023. Koejakson avulla organisaatio pystyi arvioimaan mallin sopivuutta omiin tarpeisiinsa ja tunnistamaan mahdolliset haasteet ja esteet sen käyttöönotolle

Työ pyrkii vastaamaan erityisesti seuraaviin kysymyksiin:

- Mitkä ovat kehitysprosessin Scrum-tapahtumat ja millä syklillä ne tapahtuvat?

- Miten sovelluksen laatu, turvallisuus ja ylläpidettävyys otetaan huomioon kehitysprosessin aikana?

- Miten kehitetyn prosessin toimivuutta voidaan seurata ja arvioida?

Toimeksiantajan toiveena oli, että työskentely olisi jatkossakin joustavaa ja ihmisläheistä, eli ketterän kehityksen ideologian mukaista. Tärkeää oli myös projektitiimin sitouttaminen muutokseen ja positiivisen ilmapiiriin luominen, jotta tiimin jäsenet omaksuisivat uudet toimintamallit avoimesti ja innostuneesti.

1.2 Työn toimeksiantaja

Työn toimeksiantaja Veritas Eläkevakuutus on Suomen vanhin työeläkevakuuttaja, joka tarjoaa laadukkaita eläkevakuutuspalveluja ja sijoitusratkaisuja asiakkailleen. Yhtiö on perustettu vuonna 1905, ja se on harjoittanut lakisääteistä eläkevakuutustoimintaa vuodesta 1962 asti, jolloin työntekijäin eläkelaki tuli voimaan. Vuonna 1970 voimaan astuneen yrittäjien eläkelain myötä toiminta laajeni käsittämään myös yrittäjäeläkevakuutukset. Vuonna 2022 Veritaksessa oli vakuutettuna lähes 68 000 työntekijää ja yli 14 000 yrittäjää. Kokonaisuudessaan Veritas vastaa noin 120 000 henkilön lakisääteisestä eläketurvasta. (Eläketurvakeskus 2023; Veritas 2023b; Veritas 2015b.)

Veritas pyrkii olemaan yrittäjän arvoinen eläkevakuuttaja tarjoten yksilöllistä palvelua. Yrityksessä painotetaan ihmisläheisyyttä ja asiakaslähtöisyyttä, huomioiden asiakkaiden tarpeet. Veritaksen tavoitteena on helpottaa yrittäjän arkea tarjoamalla parhaita asiakaskokemuksia sekä toimimalla yrittäjien äänenä eläkealalla. (Veritas 2023c.)

2 Ohjelmistotuotanto

Ohjelmistotuotannon tavoitteet ovat pohjimmiltaan yksinkertaiset: hallittavuus, kustannustehokkuus, ennustettavuus ja riittävä laatu (Haikala & Mikkonen 2011, 27). Software Engineering Body of Knowledge on kokoelma periaatteita ja käytäntöjä, jotka liittyvät ohjelmistotuotantoon ja ohjelmistojen kehittämiseen. SWEBOK on laadittu IEEE Computer Societyn toimesta, ja se kattaa useita tietämysalueita, kuten ohjelmiston vaatimukset, suunnittelu, rakentaminen, testaus, ylläpito, laatu, konfiguraatiohallinta ja muita tärkeitä osa-alueita ohjelmistotuotannossa. Tämä tietopohja auttaa standardoimaan ohjelmistotuotannon käsitteitä ja toimintatapoja.

SWEBOK-ohjeistuksen mukaan ohjelmistotuotanto jakautuu seuraaviin osaamisalueisiin (eng. knowledge area):

1. Vaatimusten määrittely (eng. software requirements) tarkoittaa sekä ohjelmistolle itselleen että sen onnistuneen toteuttamisen ja ylläpitämisen asettamia vaatimuksia (IEEE 2014, osa 1, 1).

2. Arkkitehtuuri (eng. software architecture) on kokonaisuus, joka sisältää järjestelmän osat, osien keskinäiset suhteet, osien ulkoiset suhteet sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja jatkokehitystä (IEEE 2022, osa 2, 1; Luukkainen & Ilves 2023).

3. Suunnittelu (eng. software design) viittaa ohjelmointiparadigmojen, komponenttien, käyttöliittymien ja muiden teknisten yksityiskohtien suunnitteluun. Sen seurauksena tulisi syntyä kuvaus sovelluksen teknisestä toteutuksesta (IEEE 2022, osa 3, 1).

4. Toteutus (eng. software construction) on nimensä mukaisesti sovelluksen toteutus edellä kuvatun teknisen kuvauksen mukaisesti ohjelmoimalla ja esimerkiksi yksikkö- ja integraatiotestausta hyödyntämällä (IEEE 2014, osa 3, 1).

5. Testaus (eng. software testing) varmistaa, että sovellus käyttäytyy odotetusti. Testaus suoritetaan dynaamisesti (eli sovellusta käyttämällä) ja se suoritetaan rajatulla määrällä ennalta määritettyjä testitapauksia. (IEEE 2014, osa 4, 1.)

6. Operatiiviset toiminnot (eng. Software engineering operations) ovat toimenpiteitä, jotka ovat tarpeen ohjelmistosovelluksen tai -järjestelmän käyttöönotossa, jatkokehityksessä ja ylläpidossa varmistaen sen eheyden ja vakauden. Tällaisia toimenpiteitä ovat esimerkiksi kohdeympäristön konfigurointi, sovelluksen seuranta ja virhetilojen hallinta. (IEEE 2022, osa 5, 1.)

7. Ylläpito (eng. software maintenance) määritellään kaikiksi toimenpiteiksi, jotka ovat tarpeen kustannustehokkaan tuen tarjoamiseksi ohjelmistolle sen ollessa käytössä (IEEE 2014, osa 5, 1).

8. Tuotteenhallinta (eng. software configuration management) ohjaa, valvoo, tunnistaa ja dokumentoi sovelluksen toiminnalliset ja fyysiset ominaisuudet, hallitsee muutoksia näihin ominaisuuksiin, kirjaa ja raportoi muutoksen käsittelyä ja toteutuksen tilaa sekä varmistaa muutosten vaatimustenmukaisuuden. (IEEE 2014, osa 6, 1.)

9. Prosessinhallinta (eng. software engineering management) voidaan määritellä kokoelmaksi toimintoja, jotka liittyvät suunnitteluun, arviointiin, mittaukseen, hallintaan, koordinointiin, johtamiseen ja riskien hallintaan ohjelmistotuotantoprojektissa. Tämän tarkoituksena on varmistaa, että ohjelmistotuotteet ja ohjelmistotuotantopalvelut toimitetaan tehokkaasti, laadukkaasti ja sidosryhmiä hyödyntäen. (IEEE 2014, osa 7, 1.)

10. Ohjelmistotuotantoprosessi (eng. software engineering process) kuvaa tapoja ja menetelmiä, joiden avulla ohjelmiston kehittäjien tulisi organisoida ja aikatauluttaa erilaisia aktiviteetteja, kuten vaatimusten määrittelyä, suunnittelua, toteutusta ja testausta ohjelmiston elinkaaren aikana (IEEE 2014, osa 8, 1).

11. Mallit ja metodit (eng. software engineering models and methods) ovat eri prosessin vaiheissa käytettyjä hyväksihavaittuja tunnustettuja toimintatapoja tai teorioita, joiden tavoitteena on tehdä ohjelmistotuotannosta systemaattista, toistettavaa ja menestyksekkäämpää (IEEE 2014, osa 9, 1).

12. Laatu (eng. software quality) voidaan määrittää esimerkiksi ohjelmiston kyvyksi täyttää sille asetetut vaatimukset ja käyttäjän tarpeet. Lisäksi osa-alue sisältää laadunvalvontaan ja laadun parantamiseen liittyviä käytäntöjä, kuten laadun arviointia. (IEEE 2014, osa osa 10, 1–2.)

13. Turvallisuus (eng. software security) on merkittävä tekijäksi sovelluskehitysprosesseissa. Turvallisen ohjelmistokehityksen periaatteena on lisätä turvallisuutta noudattamalla joukkoa vakiintuneita ja/tai suositeltuja

> sääntöjä ja käytäntöjä sekä ottamalla se osaksi jokaista kehityssyklin vaihetta. (IEEE 2022, osa 13, 1.)

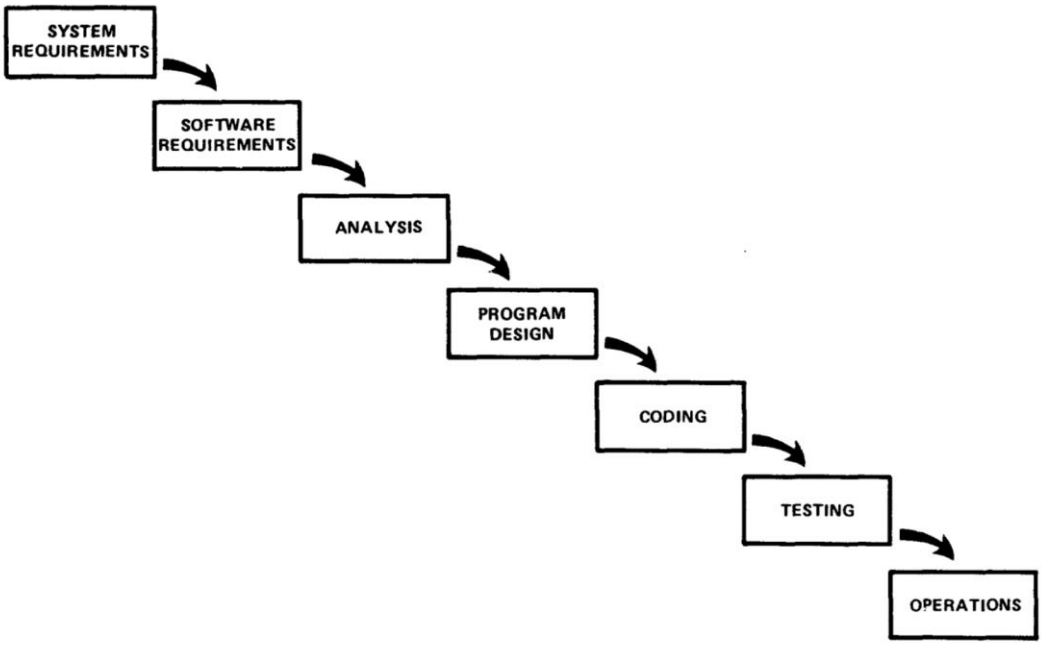
Lisäksi SWEBOK esittelee viisi ohjelmistokehittäjän sivistykseen liittyvää osaamisaluetta, jotka on rajattu työn viitekehyksen ulkopuolelle. Monet prosessin osaamisalueista, kuten turvallisuus, tapahtuvat useissa ohjelmistokehityksen elinkaaren vaiheissa ja ovat sidoksissa myös toisiin osaamisalueisiin. Vuonna 2022 julkaistu SWEBOK V4 -vedos kiinnittää erityistä huomiota turvalliseen sovelluskehitykseen ja lisää sen omaksi osaamisalueekseen. Samalla kun ohjelmistopalveluiden merkitys yritysten ja yhteiskunnan toiminnalle lisääntyy, myös ohjelmistoturvallisuuden merkitys korostuu. Kyberturvallisuuskeskuksen mukaan ”Suomessa ollaan tietoisia ohjelmistoturvallisuuden merkityksestä*”, mutta sen toteutuksen taso on vaihteleva. Ohjelmistoturvallisuuden tila 2023 -raportissa virasto peräänkuuluttaa turvallisuuden sisällyttämistä koko organisaation toimintatapoihin ja turvatyön lisäämistä kehityshankkeen alkuvaiheisiin. (Kiravuo, Timlin, Kemppainen, Eronen & Seppänen 2023, 9.)

2.1 Ohjelmistuotannon elinkaari ja elinkaarimallit

IEEE-standardi määrittää kokoelman teknisiä prosesseja, joiden avulla ohjelmisto määritellään, muunnetaan määritelmästä tuotteeksi tai palveluksi, mahdollistetaan tuotteen tai palvelun tarjoaminen ja ylläpito sekä tuotteen tai palvelun hävittäminen, kun se poistetaan käytöstä (ISO/IE/IEEE 12207 2017, 55). Nämä tekniset prosessit muodostavat ohjelmiston elinkaaren. Prosessien soveltamisen laajuus ja tiukkuus eri elinkaaren vaiheissa sekä vaiheiden kesto ja järjestys määräytyvät projektin liiketoimintatarpeiden ja teknisten tarpeiden perusteella ja valitun elinkaarimallin perusteella (ISO/IE/IEEE 24748 2018, 18). Ohjelmistotuotannon elinkaari (Software Development Life Cycle, SDLC) on se osa ohjelmiston elinkaarta, joka tähtää uuden sovelluksen tuottamiseen ja ylläpitoon. Ohjelmistokehityksen elinkaaren vaiheiksi ovat alalla yleisesti vakiintuneet vaatimusten määrittely, suunnittelu, toteutus, testaus, käyttöönotto ja ylläpito (AWS 2023; Haikala & Mikkonen 2011, 30). Koska vaiheista käytetyt termit

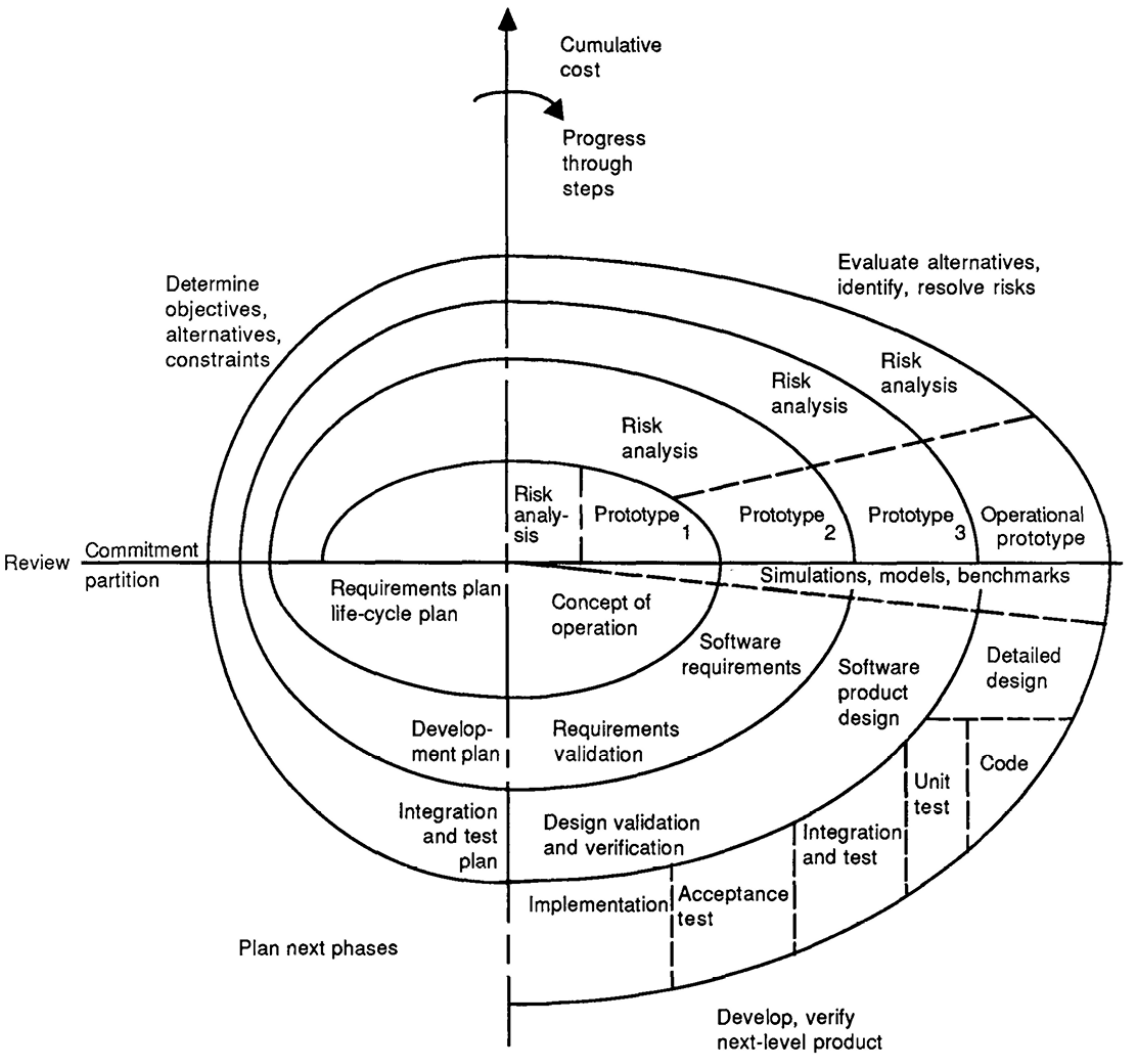
voivat vaihdella (ISO/IE/IEEE 12207 2017, 17) ja elinkaari rakenne määräytyy projektin tarpeiden perusteella, on erilaisia elinkaaria käytännössä loputon määrä.

Elinkaarimallit jakavat erityyppiset elinkaaret ryhmiksi niiden joustavuuden, toistuvuuden ja suunnittelukeskeisyyden perusteella. Ensimmäinen standardiksi muodostunut elinkaarimalli on Winston Roycen 1970-luvulla esittelemä suunnittelukeskeinen lineaarinen malli. Suoraviivaisessa mallissa elinkaaren vaiheet suoritetaan peräkkäin ja edellisen vaiheen tulee päättyä ennen siirtymistä seuraavaan vaiheeseen (kuva 1), eli vaatimukset tulee määrittää ennen suunnittelua ja toteutusta. Mallin suosio syntyi pääosin siitä, että Yhdysvaltojen puolustusministeriö (Department of Defense, DoD), joka oli tuolloin yksi maailman suurimmista ohjelmistojen tilaajista, alkoi vaatia kaikilta alihankkijoiltaan prosessin noudattamista DoD STD 2167 -standardin mukaisesti (Firesmith 1987, 1). Todellisuudessa Royce ei suositellut lineaarisia malleja ohjelmistokehityksen käyttöön, vaan totesi niiden olevan riskialttiita ja sopeutuvan huonosti muutoksiin (Royce 1970, 2). Suunnittelukeskeiset mallit osoittautuivat huonoiksi myös käytännössä, mutta ongelmista huolimatta mallien käyttö jatkui aktiivisesti standardisoinnin vuoksi aina 90-luvun puoliväliin asti (Firesmith 1987, 4–5; Larman & Basil 2003, 7).



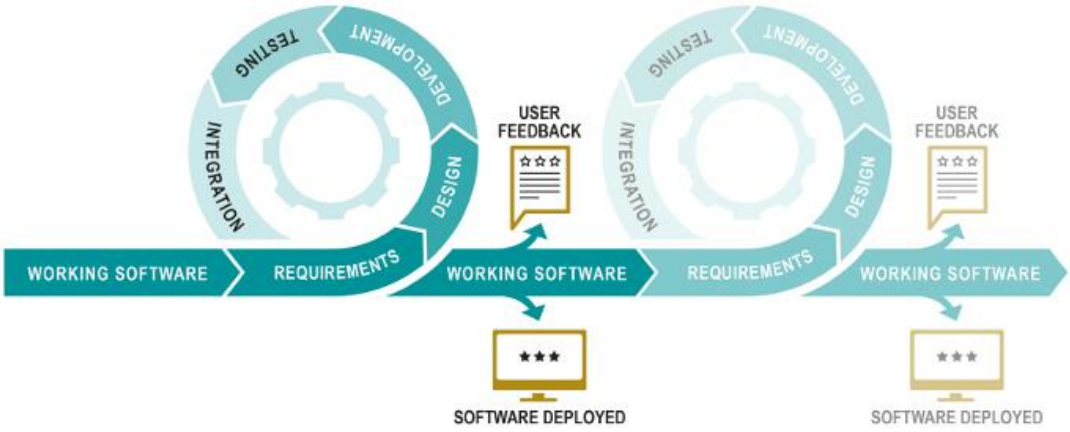
Kuva 1: Lineaarinen malli (Royce 1970)

Suunnittelukeskeisten mallien sopeutumiskyvyttömyys ja riskialttius synnytti 80- ja 90-luvuilla monia uusia ohjelmistokehitysmalleja. Näille uusille adaptiivisille malleille oli tyypillistä iterointi, rekursio ja prototyypit, joilla pyrittiin helpottamaan palautteen välittämistä käyttäjien, suunnittelijoiden ja kehittäjien välillä ja mini-moimaan muutoksiin liittyviä riskejä. Barry Boehmin julkaisema spiraalimalli (kuva 2) vakiinnutti riskipohjaisen iteraation käsitteen vuonna 1986 (Larman & Basil 2003, 6).



Kuva 2: Spiraalimalli (Boehm 1986)

Iteratiivisten ja inkrementaalisten mallien pohjalta kehittyivät lopulta erilaiset evolutiiviset mallit ja ketterät menetelmät kuten Rational Unified Process (RUP), Rapid Application Development (RAD), Extreme Programmin (XP), Scrum ja Kanban. Evolutiivisissa malleissa vaatimusten, ajan ja kustannusten arvioita muokataan säännöllisesti, kun projekti tiimin ymmärrys tuotteesta tai palvelusta kasvaa (kuva 3). Nykyiset modernit IEEE/ISO-standardit eivät enää suosittele yksittäisiä elinkaarimalleja, ja käytössä olevat elinkaaret ovat yleensä yhdistelmä erilaisia malleja ja voivat sisältää myös ketteriä metodeja (IEEE 2014, osa 8, 5; ISO/IE/IEEE 12207 2017, 18).



Kuva 3: Esimerkki evolutiivisesta kehitysmallista (GAO 2023)

Hyvin valittu, toteutettu ja hallittu elinkaari mahdollistaa ohjelmiston laadukkaan kehityksen ja ylläpidon, mikä varmistaa sen tehokkuuden ja käyttövarmuuden koko elinkaaren ajan, edesauttaa näkyvyyttä kaikille sidosryhmille, tehokkaampaa arviointia, suunnittelua ja aikataulutusta, parannettua riskienhallintaa ja kustannusarviointia sekä systemaattista ohjelmiston toimittamista ja parempaa asiakastyytyväisyyttä. (Taina 2009; AWS 2023.)

2.2 Ketterä ohjelmistokehittäminen

Ketterä ohjelmistokehitys sai alkunsa reaktiona perinteisille, raskaammille suunnitteluorientoituneille ohjelmistokehitysmenetelmille. Ketterä kehitys ei edusta yhtä tiettyä ohjelmistokehitysmallia tai standardia, vaan pikemminkin sisältää joukon arvoja ja menetelmiä, jotka edistävät ketteryyttä (ISO/IE/IEEE 12207 2017, 18). Ketterät menetelmät korostavat joustavuutta, yhteistyötä ja kykyä nopeaan reagointiin asiakasvaatimusten muuttuessa. Ketterän kehityksen alkuperäiset periaatteet ja käytännöt muotoiltiin ketterän ohjelmistokehityksen julistuksessa, joka korosti neljää keskeistä aatetta (Agile Alliance 2001):

1. yksilöt ja vuorovaikutus ennen prosesseja ja työkaluja

2. toimiva ohjelmisto ennen kattavaa dokumentaatiota

3. asiakasyhteistyö ennen sopimusneuvotteluja

4. muutokset suunnitelmiin tervetulleita.

Ketterä lähestymistapa ei rajoitu vain ohjelmistokehitykseen. Sitä sovelletaan laajalti eri aloilla, mukaan lukien projektinhallinta, liiketoiminnan kehittäminen ja tuotekehitys. Vuonna 2018 jopa 70 % yrityksistä käytti ketteriä menetelmiä projektien hallintaan (PMI 2018, 23) ja ketteryydestä on tullut olennainen osa modernia ohjelmistokehitystä ja projektinhallintaa (Hewlett Packard 2017, 1).

Ketteryyden yleistymisestä huolimatta siihen liittyy myös haasteita. Esimerkiksi Yhdysvaltojen puolustusministeriö hankintaohjeet ovat vuodesta 2010 edellyttäneet, että toimittajan tulee osallistuttaa loppukäyttäjä aikaisessa vaiheessa, pystyä tuottamaan pieniä nopeasti kehitettäviä julkaisuja, pyrkiä evolutiiviseen ja kehittämiseen ja hyödyntää modulaarista avoimen järjestelmän arkkitehtuuria (NDAA 2010). Myöhemmin tammikuussa 2020 päivitetyissä hankintaohjeissa korostettiin erityisesti ketteryyttä ja nopeutta (Lopez 2020). Vaatimuksista ja ohjeistuksista huolimatta uusimmassa Yhdysvaltain hallituksen vastuuvirasto GAO:n raportissa todetaan, ettei puolustusministeriö ole onnistunut ketterien kehitysmenetelmien käyttöönotossa (GAO 2023).

3 Scrum-mallin käyttöönotto

Ohjelmistotuotantoprosessin arviointi ja kehitys aloitettiin yhdessä kehityspäällikön kanssa keväällä 2023. Alkukartoituksessa tämänhetkisiksi konkreettisiksi ongelmakohdiksi määritettiin pitkäaikaisen suunnittelun puute ja töiden ennakoimattomuus, puutteelliset ja epäselvät määrittelyt sekä julkaisujen viivästyminen ja kasvaminen liian suuriksi kokonaisuuksiksi. Kehitysprojektissa on pyritty kerran kuukaudessa julkaistaviin versioihin, mutta käytännössä kehitystyö on edennyt lineaarisella vesiputousmallilla, ja julkaisu on tehty, kun määrittely, toteutus ja testaus on saatu valmiiksi. Mikäli jokin prosessin vaiheista on viivästynyt,

julkaisua on siirretty. Uusia määrittelyitä on voitu alkaa tuottaa vasta julkaisun jälkeen, jonka vuoksi kehitystiimi ei voi edistää testauksen aikana muita tehtäviä. Sisäisen resursoinnin lisäksi kehitysprojekti on riippuvainen ulkoisten toimittajien integraatioista ja osa julkaisuista tulee tehdä kaikissa palveluissa yhtä aikaa. Suunnitelluissa aikatauluissa pysyminen on kaikkien osapuolten työn sujumisen kannalta kriittistä.

Scrum-malli on suosituin ketterään kehitykseen suunniteltu projektinhallinnan menetelmä, joka pyrkii parantamaan tuotteen laatua, vähentämään riskejä ja sitouttamaan tiimin yhteisten tavoitteiden saavuttamiseen ja toistensa tukemiseen (digital.ai 2022; Schwaber & Sutherland 2020, 3–4). Sen etuja ovat yksinkertaisuus, läpinäkyvyys ja kyky reagoida muutoksiin (Haikala & Mikkonen 2011, 47). Kuten suurin osa nykyisin käytössä olevista ketteristä malleista, myös Scrum on yhdistelmä iteratiivisia ja lisääviä malleja (Taina 2009). Scrum sopii kehitysprojektin tarpeisiin hyvin sen nopeatempoisuuden, selkeyden ja joustavuuden ansiosta. Lisäksi malli oli ennalta tuttu kaikille projektitiimin jäsenille.

Scrum-mallissa projekti jaetaan lyhyisiin, yleensä 2–4 viikon pituisiin jaksoihin, joita kutsutaan sprinteiksi. Sprintin aikana tiimi valitsee tuoteomistajan laatimasta kehitysjonosta tärkeimmät työt tehtäväksi ja sitoutuu toimittamaan valmiin julkaisun sprintin loppuun mennessä. Jokainen päivä alkaa lyhyellä tiimitapaamisella, jossa jaetaan tietoa työn etenemisestä ja koordinoidaan tulevaa työskentelyä. Sprintin lopussa pidetään katselmointi, jossa tiimi esittelee valmiin työn tuoteomistajalle ja muille sidosryhmille. Tämän jälkeen tiimi pitää sprintin retrospektiivin, jossa arvioidaan sprintin suoritus ja mietitään parannusmahdollisuuksia seuraavaa sprinttiä varten. (Schwaber & Sutherland 2020, 5–8.)

Ulkoisten toimittajien aikataulujen yhteensovittamisen vuoksi sprintin pituudeksi valikoitui kolme viikkoa. Projektitiimin mielestä sprintin lyhyt kesto asetti haasteita erityisesti määrittelylle ja testaukselle ja voisi onnistua vain, jos näihin vaiheisiin saataisiin lisäresursseja sekä selkeät toimintaohjeet ja koulutusta.

3.1 Roolit ja vastuualueet

Scrum-mallissa on kolme keskeistä vastuualuetta. Tuoteomistaja on henkilö, joka määrittelee tuotteen vaatimukset ja asettaa prioriteetit kehitysjonolle (eng. backlog), ja varmistaa liiketoiminnan tarpeiden ymmärtämisen. Scrum Master toimii tiimin mahdollistajana ja esteiden poistajana, millä varmistetaan, että tiimi voi toimia tehokkaasti. Kehitystiimi on itseorganisoituva ryhmä, joka työskentelee tuotteen kehittämiseksi. He ottavat vastuun työstä, suunnittelevat sprinttejä ja toteuttavat niissä sovitut tehtävät. Vastuualueet muodostavat Scrum-tiimin, joka on ”yhtenäinen ammattilaisten yksikkö” ja ”joka keskittyy yhteen tavoitteeseen kerrallaan”. (Schwaber & Sutherland 2020, 5–7.)

Taulukossa 1 on kuvattu ennen Scrum-mallin käyttöönottoa tunnistettuja sisäisiä rooleja sekä arvioitu niiden sidoksia Scrum-mallin vastuualueisiin. Rooleja ja vastuualueita jatkokehitettiin koejakson aikana projektitiimin toiveiden ja tarpeiden mukaisesti. Kehitysprojektin sisäisten roolien lisäksi tunnistettiin muita sidosryhmiä, jotka ovat tiiviisti sidoksissa kehitystyössä. Tällaisia ovat mm. toimeksiantajan CRM-järjestelmästä ja integraatioista vastaavat tekniset asiantuntijat ja liiketoiminnan edustajat, jotka nostavat liiketoiminnan kehitysehdotukset toimeksiantajan sisäiselle kehitysjonolle.

Taulukko 1: Projektitiimin sisäiset roolit

Liiketoiminnan asiantuntijat	• Määrittelevät määrittämiä kehitysehdotuksia ja siirtää ne verkkopalvelun kehitysjonolle. • Vastaavat ulkoisten riippuvuuksien kuten integraatioiden tunnistamisesta. • Vastaavat kehitystiimin liiketoimintallisiin kysymyksiin.

Sovellusomistajat	• Määrittelevät ja suunnittelevat tuoteomistajien valitsemat toiminnallisuudet toteutettaviksi tehtäviksi. • Priorisoivat tehtävät kehitysjonolle.
Kehityspäällikkö	• Koordinoi töitä verkkopalvelun ja sidosryhmien välillä ja osallistuu kaikkien sidosryhmien sprinttien suunnittelupalavereihin. • Tekee tarvittaessa priorisointipäätökset eri hankkeiden välillä, jos sovellusomistajat eivät pääse yhteisymmärrykseen. • Määrittää kvartaalipalavereissa resursoinneista sekä koordinoi ulkoisten sidosryhmien kanssa yhteistyössä tehtävien toimien aikatauluttamista.
Vastaava kehittäjä	• Hoitaa Scrum Master -roolille tyypilliset tehtävät kuten tehtävien seuranta, töiden jakaminen kehitystiimin kesken ja kehitystiimin työtä hankaloittavien esteiden poistamisen. • Osallistuu määrittelyiden tuottamiseen yhdessä tuoteomistajien ja sovellusomistajien kanssa.

	• Luo testaussuunnitelman ja raportin.
Kehitystiimi	• Kehitysjonolla olevien asioiden työmäärien arviointi tarinapisteiden avulla.

Projektissa on useita päällekkäisiä rooleja. Scrum-mallin mukaan tällaista toimintamallia tulisi välttää, mutta on ymmärrettävää, ettei aina ole käytettävissä riittävästi resursseja. Päällekkäisten roolien kohdalla on tärkeää varmistaa, että henkilöllä on riittävästi aikaa kaikkien tehtävien hoitamiseen ja että henkilöllä on vain yksi tavoite kerrallaan (Crown 2023). Esimerkiksi asiakasyritykselle luodussa mallissa vastaava kehittäjä hoitaa sekä Scrum Master -roolille tyypillisiä tehtäviä, että osallistuu kehitystyöhön. Lisäksi vastaava kehittäjä osallistuu toistaiseksi myös määrittelyvaiheen dokumenttien tuottamiseen. Työt on priorisoitu siten, että vastaava kehittäjä hoitaa ensisijaisesti ne työt, jotka estävät töiden etenemisen. Täten käytännön kehitystyöt delegoidaan tarvittaessa muille tiimin jäsenille, jos niiden toteuttamiseen ei jää vastaavalla kehittäjällä riittävästi aikaa.

3.2 Sprintin tapahtumat

Toimeksiantajayrityksen toimintaa ohjataan vuosineljänneksissä. Työn organisoinnin tulee tapahtua ylhäältä alaspäin, joten myös sprinttien suunnittelu tulee aloittaa vuosineljänneksistä. Tämä suunnittelu tehdään kehityspäällikön toimesta **verkkopalvelun kvartaalipalaverissa**, johon osallistuvat myös tuotepäälliköt ja kehitystiimi. Palaverin tavoitteena on tunnistaa seuraavan kahden kvartaalin aikana toteutettavia muutoksia ja varautua näiden resursointitarpeisiin.

Tuoteomistajat nostavat kvartaalipalaverissa hyväksytyt kehitys- ja korjausehdotukset alustavian määritysten kanssa verkkopalvelun kehitysjonoon. Tuotepäälliköt ja kehitystiimi käyvät **kehitysjonon työstöpalavereissa** läpi kehitysjonolle nostetut tiketit ja määrittävät näille tarinapisteet. Jos määrittelyt muuttuvat

tai olennainen tieto lisääntyy, kehitysjonon kohteita muokataan, tarpeettomat kohdat eliminoidaan ja tarvittaessa otetaan mukaan uusia kohteita. Tikettipalaverit pidetään maanantai- ja torstai-iltapäivisin. Palaverin kesto on noin puoli tuntia.

Työn organisointi sprintteihin tapahtuu **sprintin katselmointi- ja suunnittelupalavereissa**. Palaveri pidetään viikko ennen seuraavan sprintin alkamista, testausjakson päätyttyä. Tällöin kehitystiimi siirtää mahdolliset sprintin laajuuden ulkopuolelle jäävät työt seuraavalle sprintille. Palaveriin osallistuvat kehityspäällikkö, tuotepäälliköt ja kehitystiimi.

Viikkopalaverissa koko kehitystiimi, asiakkaan tekniset asiantuntijat sekä sovellusomistajat käyvät läpi mahdolliset esteet sekä koordinoivat toteutettavia töitä eri sidosryhmien välillä. Kehitystiimi ei kokenut tarvetta päivittäisille Scrum-palavereille.

Sprintin keskivaiheilla pidettävässä **hyväksymistestauksen suunnittelupalaverissa** käydään läpi sprintin aikana toteutetut toiminnallisuudet ja nimetään kullekin testauskohteelle vastuuhenkilö. Palaveriin osallistuvat tuoteomistajat, kehitystiimi ja testaustiimi.

Itse **hyväksymistestaus** suoritetaan palaveria seuraavan viikon maanantain ja keskiviikon välisenä aikana. Testauskierroksen päätyttyä kehitystiimi arvioi ehtiikö se korjaamaan testauksen aikana havaitut kriittiset virheet seuraavaan julkaisuun mennessä vai siirretäänkö osa toiminnallisuuksista seuraavalle sprintille. Testaustulokset esitetään seuraavan sprintin suunnittelupalaverissa ja kuluvan sprintin katselmoinnissa.

Sprintin päättymisen jälkeen pidettävässä **sprintin retrospektiivissä** käydään läpi sprintin tunnuslukuja ja kerätään palautetta Scrum-tiimiltä. Palaveriin osallistuvat mahdollisuuksien mukaan kaikki sidosryhmät.

4 Laadunvalvonta

Ketterä kehitys ja Scrum-malli korostavat jatkuvaa parantamista ja laadukkaiden tulosten saavuttamista. Kehittäjien aktiivinen panostus toteutuksen laadun parantamiseen on olennainen osa näitä periaatteita. Ketterässä kehityksessä laatu nähdään prosessissa ja lopputuotoksessa, ei pelkästään testaamisen tai korjaamisen näkökulmasta. Kokonaisprosessin määrittely ja työvaiheiden selventyminen inspiroivat hyviä keskustelunaloituksia myös kehitystiimin kesken, ja kokeilujakson aikana sovittiin erinäisiä yhteisiä tekniseen toteutukseen liittyviä kriteerejä ja toteutusohjeita. Eniten tuotetun ohjelmiston laatuun vaikutti kuitenkin testauksen parantuminen.

Sovellustestauksen tavoite on varmistaa ohjelmistotuotteen laatu ja luotettavuus sekä todeta, että se täyttää sille asetetut vaatimukset, loppukäyttäjän tarpeet ja toimii odotetulla tavalla (Luukkainen & Ilves 2023). Projektissa, jossa sovellustestauksesta vastaavat liiketoimintasaajat ilman erityistä sovellustestaajan ammattitaitoa, on sekä haasteita että mahdollisuuksia. Teknisen osaamisen puute voi vaikuttaa testauksen kattavuuteen ja voi voi johtaa siihen, että joitakin ongelmia jää havaitsematta. Toisaalta henkilöiden hyvä liiketoimintaosaaminen auttaa toiminnallisuuksien tarpeenmukaisuuden toteamisessa ja on välttämätön monimutkaisten, esimerkiksi eläkelainsäädäntöön pohjautuvien toimintojen oikeellisuuden varmistamiseksi. Toimeksiantajan havaitsemia haasteita sovellustestaamiselle olivat kiire ja huonosti suunnitellut tai puutteelliset testitapaukset.

Testausprosessin hallinnointi aloitettiin peruskäsitteiden ja regressiotestien määrittelyllä. Jokaisen kolmen viikon sprintin aikana testaus- ja kehitystiimi pitävät yhdessä testauksen suunnittelupalaverin, demon ja kolmen päivän mittaisen hyväksymistestauksen. Testaussyklin tuotoksena on testausdokumentaatio, josta ilmenevät tehdyt testit, löydetyt virheet sekä virheiden kriittisyys ja luokitus. Löydettyjen virheiden tunnuslukuja seurataan sprintin jälkipalaverissa. Ensimmäisten testaussyklien aikana tunnistettiin ja määritettiin niin sanotut toistuvat regressiotestitapaukset. Seuraavana tavoitteena on näiden automaattinen testaus Robot Framework -työkalun avulla.

5 Jatkokehitysehdotukset

Roolien selkeyttämisellä, Scrum-mallilla ja testausprosessin määrittelyllä oli positiivia vaikutuksia lähes kaikkiin SWEBOK-ohjeistuksen osa-alueisiin. Kehitysprojektin hallinnassa ja sen vaiheissa olisi kuitenkin vielä runsaasti parannettavaa. Tärkeimpänä jatkokehityskohteena pidän määrittelyprosessin kehittämistä. Testausprosessin määrittely koettiin projektitiimissä erittäin hyödylliseksi ja testaustiimi toimi kokeilujakson loppupuolella jo lähes itsenäisesti. Toimeksiantajan määrittelyprosessia tulisi kehittää vastaavalla tavalla.

Toistaiseksi vaikutusten arviointi on perustunut lähinnä empiiriseen havainnointiin ja projektitiimin kokemuksiin. Scrum-mallin tunnuslukujen säännöllinen seuranta ja analysointi antaisivat organisaatiolle arvokasta tietoa siitä, miten Scrum-prosessi toimii ja miten sitä voidaan parantaa. Tunnuslukujen avulla voidaan tunnistaa ongelmakohdat, jotka saattavat estää tiimiä saavuttamasta täyttä potentiaaliaan, ja tukea jatkuvaa kehitystä kohti entistä tehokkaampaa ja tuottavampaa tuotekehitystä.

5.1 Tunnuslukujen seuranta

Scrum-menetelmässä edistymisen seurantaan käytetään useita erilaisia tunnuslukuja ja mittareita, jotka auttavat tiimiä, tuoteomistajaa ja sidosryhmiä ymmärtämään, miten projekti etenee ja kuinka hyvin seuraavat sprintit voivat onnistua.

Tarinapisteet (eng. story points) ovat mittari, joka auttaa tiimiä arvioimaan työn monimutkaisuutta, vaativuutta ja laajuutta sprintissä. Tarinapisteet eivät ole aikaan perustuva mittari vaan pikemminkin suhteellinen arvio työn vaikeudesta. Tämä auttaa tiimiä paremmin hahmottamaan ja arvioimaan tehtäviä suhteessa toisiinsa. Tämä auttaa tiimiä tekemään realistisia sitoumuksia sprintin aikana ja parantamaan arviointikykyä ajan kuluessa, kun tiimi oppii tuntemaan oman kykynsä ja työskentelynopeutensa paremmin. Tarinapisteet auttavat myös tuomaan läpinäkyvyyttä työn monimutkaisuuteen ja tarjoavat arvokasta tietoa tiimin suorituskyvystä ja kyvystä toimittaa lisäarvoa joka sprintissä.

Sprintin aikana tuotettujen toiminnallisuuksien yhteenlasketut tarinapisteet muodostavat tiimin velositeetin (eng. velocity). Mittari on yksi keskeisimmistä Scrum-mittareista ja auttaa ennustamaan, kuinka monta tehtävää tiimi voi toteuttaa tulevissa sprinteissä tai mikä on tietyn kokoisen tehtävän arvioitu kesto. Luotettavan velositeetin määrittyminen kestää yleensä useita sprinttejä. Sprinttien suunnittelun kannalta keskeisiä tunnuslukuja ovat jäljellä olevan työn määrä (eng. burndown) ja arvioitujen työmäärien suhde toteutuneisiin työmääriin (eng. burnup).

5.2 Määrittelyprosessin kehittäminen

Kuten kehitystyö, myös määrittely on toteutettu kehitysprojektissa lineaarisena toimenpiteenä, jossa koko toiminnallisuus tai palvelu on pyritty kuvaamaan mahdollisimman yksityiskohtaisesti alusta loppuun ennen toteutusta. Käytännössä määrittelyt ovat aina muuttuneet kehitysprosessin myöhemmissä vaiheissa. Lisäksi määrittelyt ja vaatimukset on koettu kehitystiimin osalta puutteellisiksi, eivätkä määrittelyprosessiin osallistuvat tahot tiedä, mitä tai miten heidän tulisi määritellä.

Määrittelyprosessin selkeyttäminen tulisi aloittaa jakamalla prosessi vaiheisiin. Prosessin eri vaiheille tulee olla nimetyt vastuuroolit, sovitut tuotosdokumentit ja määrittelydokumentaatiota täydennetään jokaisessa vaiheessa sovitusti. Lineaarisesti etenevän prosessin sijaan määrittely tehdään spiraalimallisesti ja täydentäen. Onnistuneella määrittelyllä voitaisiin merkittävästi parantaa kehitystyön sujuvuutta sekä lisätä lopputuotteen laatua.

6 Yhteenveto

Ensimmäisen puolen vuoden aikana projektitiimi suoritti onnistuneesti aikataulussa seitsemän kolmen viikon mittaista sprinttiä, ja lähes kaikki toimeksiantajan toivomat toiminallisuudet saatiin toimitettua onnistuneesti. Myös parannus tuotetun tuotteen laadussa oli selvä. Erään lokakuussa päättyneen projektin palautteessa todettiin, ettei julkaisua seuranneen kuukauden aikana ilmennyt yhtään

virhettä. Uuden mallin käyttöönotossa koettiin kuitenkin myös haasteita. Nopeampi työrytmi ja kankeammat palaverimallit koettiin ahdistaviksi ja aikaisempaa "rentoa jutustelua" jäätiin kaipaamaan. Syksyn koejakson jälkeen sprintin pituutta päätettiin pidentää 4 viikkoon ilman, että sprintille nostettavien tehtävien määrä lisääntyy. Tällä haluttiin varmistaa, että projektitiimille jää tarpeeksi aikaa määrittelyyn ja testaukseen. Myös rooleihin ja vastuualueisiin tehtiin koejakson aikana muutoksia, jotka mahdollistivat töiden tasaisemman jakautumisen. Uudet vastuualueet otettiin kehitystiimissä hienosti vastaan ja koettiin myös mielekkäiksi.

Projektitiimin ja toimeksiantajan lisäksi työstä oli merkittävää etua myös kirjoittajan oman osaamisen lisäämiselle. Työn aikana hankittu uusi tieto ja näkökulmat antoivat mahdollisuuden soveltaa parhaita käytäntöjä ja toimintatapoja käytännön tilanteissa. Erityisesti Scrum Masterin roolin omaksuminen vaati syventymistä ketterän kehityksen periaatteisiin ja Scrum-mallin käytäntöihin. Tietämys lisäsi uskoa omiin päätöksiin ja helpotti projektitiimin tehokasta johtamista. Työn edetessä oli ilo huomata, miten uusi tieto integroitui saumattomasti aiempaan kokemukseen ja toi lisäarvoa roolissa onnistumiselle. Työn myötä syntyi myös selkeämpi ymmärrys omasta osaamisesta ja vahvuuksista. Kaiken kaikkiaan työ ei ainoastaan edistänyt käytännön taitoja ja tiedon lisäämistä, vaan myös tuki henkilökohtaista ja ammatillista kehitystä. Se vahvisti käsitystä siitä, että jatkuva oppiminen ja oman osaamisen kehittäminen ovat olennainen osa menestyksekästä työuraa.

Lähteet

Agile Alliance 2001. Ketterän ohjelmistokehityksen julistus. URL: https://agilemanifesto.org/iso/fi/manifesto.html. Luettu: 29.10.2023.

AWS 2023. What is SDLC? - Software Development Lifecycle Explained. URL: https://aws.amazon.com/what-is/sdlc/. Luettu: 21.10.2023.

Boehm, B.W. 1986. A spiral model of software development and enhancement. Computer, 21, s. 61–72. URL: https://api.semanticscholar.org/CorpusID:1781829.

Crown, G. 2023. Can one person fill multiple roles on a Scrum Team? | Scrum.org. URL: https://www.scrum.org/resources/blog/can-one-person-fill-multiple-roles-scrum-team. Luettu: 27.2.2024.

digital.ai 2022. 16th State of Agile Report.

Eläketurvakeskus 2023. Eläketurvan muutokset vuosi vuodelta. URL: https://www.etk.fi/suomen-elakejarjestelma/elakeuudistukset/aiemmat-uudistukset/muutokset-vuosi-vuodelta/. Luettu: 11.10.2023.

Firesmith, D.G. 1987. Should the DOD mandate a standard software development process? URL: https://www.researchgate.net/publication/255680192.

GAO 2023. Software Acquisition: Additional Actions Needed to Help DOD Implement Future Modernization Efforts. URL: https://www.gao.gov/products/gao-23-105611. Luettu: 29.10.2023.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Talentum Media Oy.

Hewlett Packard 2017. Agile is the new normal: Adopting Agile project management.

IEEE 2014. SWEBOK V3: Guide to the software engineering body of knowledge. IEEE Computer Society.

IEEE 2022. SWEBOK V4 Beta: Guide to the software engineering body of knowledge (DRAFT 31.12.2022).

ISO/IE/IEEE 12207 2017. Systems and software engineering - Software life cycle processes. URL: https://doi.org/https://doi.org/10.1109/IEEESTD.2017.8100771.

ISO/IE/IEEE 24748 2018. Systems and software engineering - Life cycle management - Part 1: Guidelines for life cycle management. URL: https://doi.org/https://doi.org/10.1109/IEEESTD.2018.8526560.

Kiravuo, T., Timlin, P., Kemppainen, K., Eronen, J. & Seppänen, S. 2023. Ohjelmistoturvallisuuden tila 2023 Nykytilaraportti.

Larman, C. & Basil, V.R. 2003. Iterative and Incremental Development: A Brief History.

Lopez, C.T. 2020. Defense Department Seeks to Achieve Agile, Adaptive Acquisition. U.S. Department of Defense.

Luukkainen, M. & Ilves, K. 2023. Ohjelmistotuotanto 2023. URL: https://ohjelmistotuotanto-hy.github.io/. Luettu: 22.10.2023.

NDAA 2010. SEC. 804. IMPLEMENTATION OF NEW ACQUISITION PROCESS FOR INFORMATION TECHNOLOGY SYSTEMS. URL: https://www.wifcon.com/dodauth10/dod10_804.htm. Luettu: 29.10.2023.

PMI 2018. Pulse of the Profession 2018: Success in Disruptive Times.

Royce, W.W. 1970. Management of the development of Large Software.

Schwaber, K. & Sutherland, J. 2020. Scrum Opas.

Taina, J. 2009. Ohjelmistoprosessit ja ohjelmistojen laatu. Kevät 2009, luentokalvot. URL: https://www.cs.helsinki.fi/u/taina/opol/k-2009/. Luettu: 22.10.2023.

Veritas 2023a. Verkkopalvelun käyttö. URL: https://veritas.fi/sivuston-kayttoehdot/verkkopalvelun-kaytto/. Luettu: 11.10.2023.

Veritas 2015a. Helpompaa elämää ja hyvinvointia. URL: https://veritas.fi/vuosikertomus2014/asiakkaat/#helpompaa-elamaa-ja-hyvinvointia. Luettu: 11.10.2023.

Veritas 2023b. Vuosikertomus. URL: https://veritas.fi/tietoa-meista/vuosikertomus/. Luettu: 11.10.2023.

Veritas 2015b. Vuosikertomus 2014. URL: https://veritas.fi/vuosikertomus2014/veritas-elakevakuutus-2014/. Luettu: 11.10.2023.

Veritas 2023c. Visiot ja arvot. URL: https://veritas.fi/tietoa-meista/visio-ja-arvot/. Luettu: 11.10.2023.