Mikko Ryynänen

TURVALLISUUDEN SUUNNITTELU KETTERÄSSÄ JÄRJESTELMÄKEHITYKSESSÄ

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2015

TIIVISTELMÄ

Ryynänen, Mikko
Turvallisuuden suunnittelu ketterässä järjestelmäkehityksessä
Jyväskylä: Jyväskylän yliopisto, 2015, 88 s.
Tietojärjestelmätiede, Pro gradu -tutkielma
Ohjaajat: Leppänen Mauri; Siponen Mikko

Tietojärjestelmien kehittymisen myötä myös niiden turvallisuus on noussut yhä tärkeämpään rooliin. Turvallisuuden suunnitteluun on kehitetty useita menetelmiä ja standardeja, joiden avulla tietojärjestelmien turvallisuusnäkökohdat voidaan kattavasti huomioida. Monet tällaiset menetelmät toimivat parhaiten vakaassa toimintaympäristössä, jossa kehitettävän järjestelmän turvallisuus- ja toiminnalliset vaatimukset voidaan suunnitella kattavasti jo ennen järjestelmän varsinaista toteuttamista. Ketterän järjestelmäkehityksen menetelmät sen sijaan korostavat toiminnallisuutta jo varhaisessa vaiheessa. Kattavan suunnittelun ja dokumentaation sijaan painopiste on iteratiivisessa kehittämisessä ja muutokset ovat mahdollisia myös myöhemmässä vaiheessa kehitystyötä. Tämä voi asettaa haasteita turvallisuuden suunnittelulle, jonka vaatimukset suhteessa ketterään kehittämiseen voivat olla osittain ristiriitaisia. Tässä tutkielmassa selvitettiin, mitä turvallisen tietojärjestelmän kehittämisstandardeja ja -menetelmiä on esitetty käytettäväksi ja miten turvallisuutta voidaan suunnitella osana ketterää järjestelmäkehitystä. Tutkimusta varten luotiin teoreettinen viitekehys, johon perustuen tarkasteltiin useita aihepiirin julkaisuja ketterän kehittämisen ja turvallisuuden suunnittelun integroinnista prosessin, roolien, käytänteiden ja tuotoksien näkökulmasta. Tutkimuksen tuloksena tunnistettiin yleisimmät ketterän kehittämisen ja turvallisuuden suunnittelun integroinnin prosessimallit ja roolit. Lisäksi esiteltiin lukuisia integrointiin soveltuvia käytänteitä ja niiden tuotoksia. Tutkimuksen tuloksia voidaan hyödyntää esimerkiksi korkeaa turvallisuutta vaativissa ketterissä järjestelmäkehityksen projekteissa. Toisaalta tuloksia voidaan hyödyntää myös yleisesti tietojärjestelmien kehittämisessä, sillä yhä useammassa projektissa turvallisuus tulee ottaa jollain tasolla huomioon. Tutkimusalue on suhteellisen laaja, ja tutkielmassa esitetään myös mahdollisia jatkotutkimusaiheita.

Asiasanat: turvallisuuden suunnittelu, tietoturvallisuus, ketterä järjestelmäkehitys, Scrum, XP

ABSTRACT

Ryynänen, Mikko
Security Engineering in Agile Systems Development
Jyväskylä: University of Jyväskylä, 2015, 88 p.
Information Systems Science, Master's thesis
Supervisors: Leppänen Mauri; Siponen Mikko

Information systems are evolving rapidly and the importance of security issues is growing. There are many security engineering methods and standards, which address these security issues. Many of these methods and standards are suitable for stable environments, where security and functional requirements can be specified extensively before the actual implementation. Agile methods, however, emphasize functionality in the early stages of development. They highlight iterative development and changing requirements instead of extensive documentation and planning. This is challenging from the perspective of security engineering, for the reason that some of the security engineering practices contradict to agile development. The aim of this thesis is to find out, what methods have been proposed to address security engineering in agile systems development. More specifically, this thesis focuses on the integration of agile development and security engineering. A theoretical framework was created for this study, forming the basis for analyzing the process, roles, practices and artefacts proposed in studies on the integration. As a result of this study, common process models and roles of security engineering and agile development integration were recognized. In addition, a number of practices and artefacts advancing agile security engineering were introduced. The results of this thesis can be used in secure critical systems development projects. On the other hand, security is increasingly important in any systems development project and the results of this study can be also applied there. The research topic of this thesis is relatively wide and further research themes are also proposed.

Keywords: security engineering, information security, agile systems development, Scrum, XP

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Turvallisuus on tärkeä osa tietojärjestelmien kehittämistä. Tietojärjestelmien turvallisuuspuutteet voivat johtaa esimerkiksi luottamuksellisten tietojen luvattomaan käyttöön tai paljastamiseen, ja pahimmillaan ne voivat estää koko järjestelmän käyttämisen. Aihetta on käsitelty alan kirjallisuudessa pitkään, ja ensimmäiset julkaisut löytyvät jo vuosikymmenten takaa. Esimerkiksi Anderson (1972) käsittelee tietokoneiden turvallisuutta sotilaallisessa ympäristössä, Parker (1973) esittää tietokoneiden hyväksikäytön tutkimuksen ja Hoffman (1977) puolestaan käsittelee aikansa moderneja menetelmiä tietoturvallisuuden suunnitteluun. Nykypäivänä tietojärjestelmien turvallisuus on noussut huomattavasti suurempaan merkitykseen, kun organisaatiot ovat yhä enemmän riippuvaisia tietojärjestelmistä jopa strategisena kilpailuetuna (Kankanhalli, Teo, Tan, & Wei, 2003). Samalla kun tietojärjestelmien käyttö kasvaa, myös vaatimukset niiden turvallisuudelle kasvavat (Mouratidis, Giorgini, & Manson, 2003).

Tietojärjestelmien turvallisuudessa keskeistä on ohjelmistojen turvallisuus ja turvallisten ohjelmistojen kehittäminen (McGraw, 2004). Myös ohjelmistokehitys on vuosien saatossa muuttunut. Vesiputousmalli (Royce, 1970) edustaa perinteistä näkemystä ohjelmistokehityksestä, jossa kehitys tapahtuu vaiheittain ja toiminnallisuus rakennetaan vasta kehityksen myöhemmissä vaiheissa. Mallin tilalle on noussut 2000-luvulla ketterä lähestymistapa (engl. agile approach), jonka arvot ja periaatteet on esitetty Agile-manifestissa (Agile Alliance, 2001). Ketterän ohjelmistokehityksen menetelmille on ominaista muun muassa iteratiivisuus, inkrementaalisuus ja nopea reagointi muutoksiin. Pitkälle ennakoidun suunnittelun sijaan ketterässä lähestymistavassa painotetaan järjestelmien toiminnallisuuksien toteuttamisen aloittamista jo varhaisessa vaiheessa ohjelmistokehitystä.

Vaikka turvallisuuden suunnittelu ja ohjelmistokehitys ovat vuosikymmenten saatossa uudistuneet, ovat ohjelmistojen turvallisuuspuutteet edelleen yleisiä ja ongelma on yhä kasvamassa (McGraw, 2004). Ohjelmistokehityksessä turvallisuusvaatimuksia käsitellään ei-toiminnallisina vaatimuksina (Chung & Nixon, 1995), mutta usein ne määritellään vasta itse järjestelmän määrittelyn jälkeen (Mouratidis ym., 2003). Tämä voi puolestaan aiheuttaa monia ongelmia

kuten ristiriitaisia vaatimuksia, järjestelmän toiminnallisia ongelmia ja lisääntyneitä kustannuksia (Baskerville, 1992).

Yleisesti hyväksyttyä ja laajasti käytettyä menetelmää turvallisuuden suunnittelun ja järjestelmäkehityksen yhdistämisestä ei ole esitetty. Turvallisuuden suunnittelu voi muodostua erityisen haasteelliseksi ketterässä lähestymistavassa, jossa pääpaino on usein järjestelmän toiminnallisissa ominaisuuksissa. Olemassa olevissa ketterän kehittämisen menetelmissä ei itsessään ole huomioitu turvallisuuteen liittyviä näkökohtia (Ge, Paige, Polack & Brooke, 2007). Sen sijaan turvallisuuden suunnittelun ja ketterän järjestelmäkehityksen yhdistämiseen on ehdotettu monia integrointiesityksiä ja käytänteitä. Esimerkiksi Singhal ja Singhal (2011) esittävät koko järjestelmän kehitysprosessin kattavaa ketterän turvallisuuden viitekehystä. Baca ja Carlsson (2011) analysoivat olemassa olevia turvallisuuden suunnittelun aktiviteetteja ja esittävät niiden hyödyntämistä soveltuvin osin ketterässä kehittämisessä. Peeters (2005) puolestaan räätälöi ketterässä kehittämisessä yleisesti käytettyjä käyttäjäkertomuksia paremmin turvallisuuden suunnitteluun soveltuviksi. Vaikka integrointiin on ehdotettu monia ratkaisuja, lähestyvät useimmat niistä kuitenkin aihetta vain kehittämisen tietystä osa-alueesta. Toisaalta laajemmin integrointia käsittelevät esitykset ovat usein yleisluontoisia, eikä mikään yksittäinen menetelmä ole saanut laajempaa suosiota käytännön kehitystyössä.

Tämän tutkielman tarkoituksena on selvittää, millä tavalla turvallisuuteen liittyvät piirteet voidaan huomioida ketterässä järjestelmäkehityksessä. Tutkimusongelmana on:

Miten turvallisuutta voidaan suunnitella osana ketterää järjestelmäkehitystä?

Tutkimusongelman kannalta keskeisiä kysymyksiä ovat:

- Millaisia menetelmiä ja standardeja on esitetty turvallisten tietojärjestelmien suunnitteluun?
- Mitä haasteita ja hyötyjä turvallisuuden suunnittelun integroimiseen ketterään ohjelmistokehitykseen liittyy?
- Millaisia menetelmiä ja käytänteitä on esitetty turvallisuuden suunnitteluun osana ketterää kehittämistä?

Tutkimusongelmaa tarkastellaan alan kirjallisuuteen perustuen käsitteellisteoreettisella tutkimusotteella. Relevanttia kirjallisuutta haetaan ja valitaan soveltaen integroivan kirjallisuuskatsauksen menetelmää (Torraco, 2005). Menetelmän tarkoituksena on katselmoida monipuolisesti aikaisempaa aineistoa ja tuottaa kriittisen sekä aiempaa laajemman tarkastelun avulla uutta tietoa aihealueesta (Torraco, 2005). Integroivaa kirjallisuuskatsausta sovelletaan alkaen aineiston keräämisestä sähköisistä tietokannoista ja niiden karsinnasta eri vaiheissa. Aineisto analysoidaan sisällöllisesti ja tarkastelun perusteella esitetään lopulta johtopäätökset. Sisällöllisessä analyysissä käytetään apuna myös tutki-

muksen teoreettista viitekehystä, jonka avulla aineistoa jäsennellään eri osa-alueisiin.

Tutkielman lähdeaineistoksi on rajattu viitekehykseen perustuen julkaisut, jotka huomioivat ketterän kehittämisen ja turvallisuuden suunnittelun integroinnin prosessin tai jokin sen osa-alueen, roolit ja tarvittavat käytänteet sekä tuotokset. Yhdeksän laadullisen analyysin ja viitekehyksen tarkastelun perusteella valittua integrointiesitystä kuvataan yksityiskohtaisemmin. Tutkielmassa esitettyjä useita käytänteitä ja menetelmiä voidaan soveltaa monipuolisesti erityisesti turvallisuuskriittisessä järjestelmäkehityksessä, mutta myös yleisesti käytännön kehitystyössä.

Tutkielma koostuu kuudesta luvusta. Luvussa 2 määritellään tarkastelun perustaksi tietojärjestelmien turvallisuuteen liittyviä käsitteitä ja esitellään kaksi yleistä turvallisuuden standardia, SSE-CMM (ISO/IEC 21827, 2008) ja CC (CC, 2012) sekä kaksi käytäntöä. Lisäksi esitellään käyttötapauksien räätälöity versio, väärinkäyttötapaus (McDermot ja Fox, 1999) ja hyökkäyspuut (Schneier, 1999). Luvussa 3 käydään läpi ketterän kehittämisen määritelmä ja kahden suositun ketterän menetelmän, XP:n (Beck, 1999b; Beck & Andres, 2004) ja Scrumin (Schwaber & Beedle, 2002; Schwaber & Sutherland, 2013) prosessit, roolit ja käytänteet. Luvussa 4 keskitytään turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämiseen. Luvun aluksi tarkastellaan integroinnissa yleisesti esiintyviä haasteita ja hyötyjä, jonka jälkeen esitellään tutkielmassa käytetty tutkimusprosessi ja useita integraation ratkaisuehdotuksia. Integrointiesityksistä kuvataan niiden käytänteet, roolit sekä kehittämisprosessi tai sen osa-alue. Luvussa 5 kuvataan integrointiesitysten tarkastelua varten luotu viitekehys ja vertaillaan integrointiratkaisuja viitekehykseen perustuen. Lopuksi esitetään tutkielman pohdinta ja yhteenveto.

2 TURVALLISEN TIETOJÄRJESTELMÄN KEHITTÄMISSTANDARDEJA JA -MENETELMIÄ

Seuraavissa luvuissa kuvataan aihepiirin tarkastelun kannalta keskeisimmät tietojärjestelmien turvallisuuden käsitteet, esitellään kaksi yleistä tietoturvastandardia ja arviointimenetelmää, SSE-CMM (ISO/IEC 21827, 2008) ja CC (CC, 2012) sekä käydään läpi lyhyesti niihin pohjautuvia ohjelmistokehityksen sovelluksia. Lisäksi esitellään kaksi turvallisuuden suunnittelun käytännettä, joista ensimmäinen on käyttötapauksien erikoistapaus, väärinkäyttötapaukset (engl. abuse case) (McDermot ja Fox, 1999) ja toinen järjestelmään kohdistuvia uhkia mallintava hyökkäyspuu (engl. attack trees) (Schneier, 1999). Luvun tarkoituksena on luoda käsitteellinen perusta aihepiirin tarkastelulle ja antaa esimerkkejä turvallisuuden suunnittelun standardeista ja menetelmistä.

2.1 Tietojärjestelmien turvallisuuteen liittyviä käsitteitä

ISO/IEC 17799 (2005) –standardin mukaan *tietoturvallisuudella* (engl. information security) tarkoitetaan tiedon luottamuksellisuuden (engl. confidentiality), yhtenäisyyden (engl. integrity) ja saatavuuden (engl. availability) säilyttämistä. Lisäksi siihen voi liittyä muita ominaisuuksia kuten aitous (engl. authencity), vastuullisuus (engl. accountability), kiistämättömyys (engl. non-repudiation) ja luotettavuus (engl. reliability) (ISO/IEC 17799, 2005). ITIL (2011) kuvaa saatavuutta tiedon käytettävyydellä ja saatavuudella tarvittaessa, luotettavuutta tiedon tutkimisella tai paljastamisella vain niille, joilla on siihen oikeus ja yhtenäisyyttä tiedon täydellisyydellä, tarkkuudella ja luvattoman käytön suojauksella. Koska ISO/IEC 17799 (2005) –standardin määritelmä on yleisesti viitattu tietojärjestelmien turvallisuutta käsittelevässä kirjallisuudessa (esim. Pearson & Yee, 2013; Posthumus & Von Solms, 2004; Ma, Johnston & Pearson, 2008), käytetään sitä käsitteellisenä perustana tämän tutkielman *tietojärjestelmien turvallisuudelle*. Lisäksi määritelmää täydennetään ITIL:in (2011) tietoturvallisuuden kuvauksella.

Tutkielman keskeisenä osa-alueena on myös ohjelmistokehitys ja ohjelmistojen turvallisuus. *Ohjelmistojen turvallisuudella* tarkoitetaan McGrawn (2004) mukaan turvallisen ohjelmiston suunnittelua, ohjelmiston turvallisuuden varmistamista ja kehittäjien, ohjelmistoarkkitehtien sekä käyttäjien kouluttamista turvallisten ohjelmistojen tuottamiseen. Määritelmä sisältää olennaiset osat turvallisesta ohjelmistokehityksestä ja sisältää paitsi teknisen-, myös sosiaalisen sekä sosio-teknisen näkökulman, joiden mukaan turvallisuus ei koostu ainoastaan teknisestä kehittämisestä. Vaikka McGraw (2004) lisää ohjelmistojen turvallisuuteen kuuluvaksi myös kehitystyön jälkeisen ajan, rajataan tässä tutkielmassa tarkasteluun ainoastaan varsinainen kehitystyö ja turvallisuuden suunnittelu osana ohjelmistokehitystä.

Ohjelmistojen turvallisuuden yhteydessä esiintyy usein käsite *turvallisuuskriittinen ohjelmisto* (engl. security-critical software), jota käytetään yleisenä terminä ohjelmistoille, joissa turvallisuuteen tulee kiinnittää erityistä huomiota. Ihmisiin liittyen turvallisuuskriittisellä (engl. safety-critical) ohjelmistolla tarkoitetaan IEEE 1228 (1994) -standardin mukaan ohjelmistotuotteita, joiden toimimattomuudesta voi olla seurauksena hengen menetys, vakava haitta tai laajalle levinnyt negatiivinen sosiaalinen vaikutus.

Turvallisuuden varmistaminen (engl. assurance) määritellään ISO/IEC 21827 (2008) -standardin mukaan luottamuksen perustaksi sille, että toimitus täyttää sille asetetut turvallisuustavoitteet. Ohjelmistokehityksessä turvallisuuden varmistaminen voidaan nähdä luottamuksen tasona siitä, että ohjelmiston toiminnallisuudet ovat tarkoituksenmukaisia ja ohjelmistossa ei ole tarkoituksella tai vahingossa suunniteltuja haavoittuvuuksia (Farroha & Farroha, 2011). Määritelmä sopii hyvin myös yleisesti järjestelmäkehitykseen.

Turvallisuuden suunnittelulla pyritään muun muassa ymmärtämään organisaation turvallisuusriskit, tuomaan esille riskeihin liittyvät turvallisuustarpeet ja muuttamaan ne muiden tieteenalojen aktiviteeteiksi sekä luomaan luottamus turvallisuusmekanismien toimivuuteen (ISO/IEC 2187, 2008). Määritelmää voidaan soveltaa turvallisuuden suunnitteluun yleisesti, mutta ohjelmistokehityksen ja turvallisten tietojärjestelmien yhteydessä turvallisuusriskeillä ja -tarpeilla voidaan käsittää muun muassa kehitettävään järjestelmään ja sen ympäristöön liittyviä riskejä. Aktiviteetit sisältävät esimerkiksi tarvittavien turvallisuusmekanismien kehittämisen tehtäviä, ja luottamukseen puolestaan kuuluu mekanismien toimivuuden varmistaminen. Jälkimmäinen tarkoittaa esimerkiksi järjestelmän testausta tai dokumentaatiota toteutetuista mekanismeista.

Tietojärjestelmien turvallisuus on aiheena laaja sisältäen myös lukuisia muita käsitteitä. *Riski* määritellään yleisenä käsitteenä ISO/IEC (2009) -standardin mukaan mahdollisiksi tapahtumiksi ja niiden seurauksiksi tai niiden yhdistelmäksi. Tietojärjestelmien yhteydessä riskillä tarkoitetaan Stoneburnerin, Goguen ja Feringan (2002) mukaan haavoittuvuuksia hyödyntävien uhkien ja niiden seurausten todennäköisyyksiä. *Uhkalla* (engl. threat) tarkoitetaan vastustajan valmiuksia, aikomuksia ja hyökkäysmenetelmiä tai mitä tahansa sisäistä tai ulkoista tapahtumaa, joka voi aiheuttaa haittaa tiedolle, järjestelmälle tai ohjelmalle, tai saada ne aiheuttamaan haittaa muille (ISO/IEC 21827, 2008). *Haa-*

voittuvuus (engl. vulnerability) tarkoittaa Stoneburnerin ym. (2002) mukaan virhettä tai heikkoutta järjestelmän turvallisuuden proseduureissa, suunnittelussa, toteutuksessa tai sisäisessä ohjauksessa, jota voidaan käyttää vahingossa tai tarkoituksella ja joka johtaa turvallisuusrikkomukseen tai järjestelmän turvallisuuspolitiikan rikkomiseen.

2.2 SSE-CMM

SSE-CMM (engl. Systems Security Engineering – Capability Maturity Model) (ISO/IEC 21827, 2008) on yleinen tietoturvastandardi ja turvallisuuden suunnittelun kypsyysmalli, jonka kehitystyössä on ollut mukana yli 50 organisaatiota eri maista. Erityisesti malliin ovat vaikuttaneet Australia, Kanada ja Yhdysvallat sekä Eurooppa. SSE-CMM ei esitä tiettyä prosessia tai menetelmää turvallisuuden suunnitteluun, vaan sen tarkoituksena on tarkastella turvallisuuden suunnittelun prosesseja ja erityisesti niiden kypsyyttä. Malli on tarkoitettu paitsi organisaatioiden turvallisuuskäytäntöjen arviointiin ja niihin liittyvien parannusten määrittämiseen, myös organisaation eri sidosryhmien, kuten asiakkaiden tai turvallisuuden arvioijien menetelmäksi. (ISO/IEC 21827, 2008.)

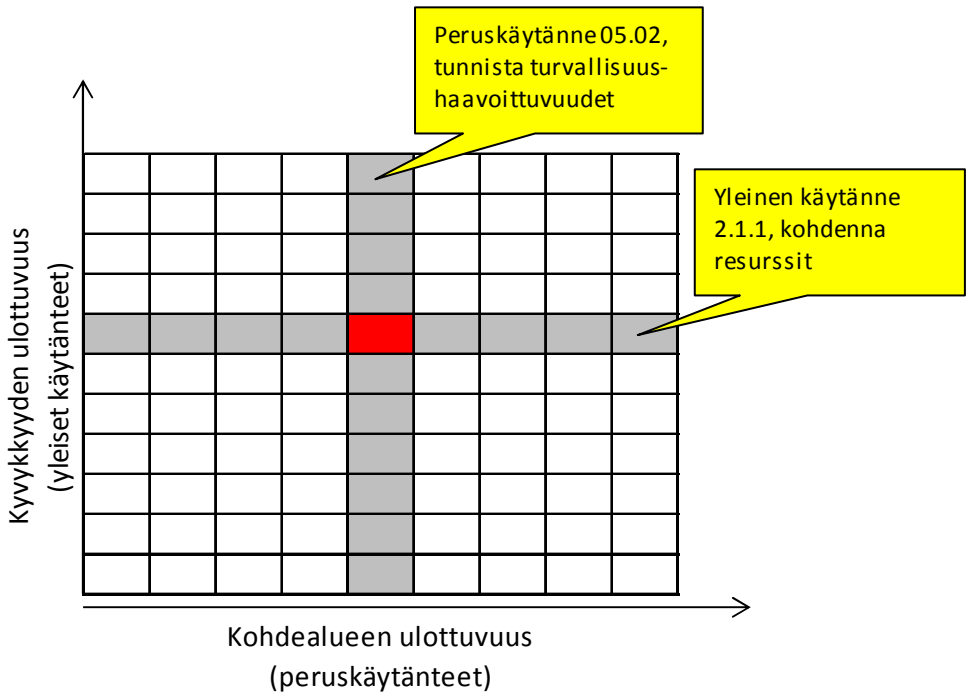
SSE-CMM kattaa turvallisuuden suunnittelun seuraavat osa-alueet (ISO/IEC 21827, 2008):

- tuotteen koko elinkaaren kehitystyöstä käyttöönottoon, ylläpitoon ja käytöstä poistamiseen
- organisaation aktiviteetit, sisältäen johdon, organisaation ja kehittämisen alueet
- riippuvuudet toisiin osa-alueisiin kuten järjestelmä-, sovellus- ja laitteistokehittämiseen sekä järjestelmänhallintaan ja ylläpitoon
- vuorovaikutuksen toisiin organisaatioihin, sisältäen hankintatoimen, järjestelmänhallinnan ja sertifiointitoiminnan.

Vaikka SSE-CMM ei ehdota käytettäväksi tiettyä turvallisuuden suunnittelun prosessia tai menetelmää, tulee organisaatiolla kuitenkin olla jokin prosessi joka sisältää mallissa esitetyt turvallisuuden käytänteet. (ISO/IEC 21827, 2008.)

SSE-CMM sisältää kaksi ulottuvuutta, jotka ovat *kohdealue* (engl. domain) ja *kyvykkyys* (engl. capability), joiden tarkoituksena on erottaa selkeästi turvallisuuden suunnittelu ja johtamistoiminta toisistaan. Kohdealueen ulottuvuus sisältää turvallisuuden suunnittelussa tarvittavat käytänteet, jotka on mallissa nimetty *peruskäytänteiksi* (engl. base practices). Peruskäytänteitä on yhteensä 129, ja ne on jaettu edelleen 22 *prosessialueeseen* (engl. process areas). Kyvykkyyden ulottuvuus sisältää organisaation käytänteet kuten prosessin hallinnan, ja siinä suoritettavat aktiviteetit on nimetty *yleisiksi käytänteiksi* (engl. generic practices). Ulottuvuudet liittyvät toisiinsa siten, että yleiset käytänteet sisältävät aktiviteetit jotka tulisi suorittaa osana peruskäytänteiden toteuttamista.

(ISO/IEC 21827, 2008.) Alla olevassa kuviossa (kuvio 1) on esitetty ulottuvuuksien keskinäinen suhde yhden peruskäytänteen, järjestelmän turvallisuushaavoittuvuuksien tunnistamisen, osalta. Jotta peruskäytänne voidaan toteuttaa, on sille osoitettava riittävät henkilöstö- tai muut resurssit. Peruskäytänteen ja yleisen käytänteen yhdistämisellä saadaan hyvä kuva siitä, millä tasolla organisaatio suoriutuu tietystä aktiviteetista. (ISO/IEC 21827, 2008.)



KUVIO 1 SSE-CMM ulottuvuudet ja niiden liittyminen toisiinsa (ISO/IEC 21827, 2008, s. 23)

Kun tarkastellaan kohdealueen ulottuvuutta tarkemmin, jakaa SSE-CMM prosessialueet ja peruskäytänteet kahteen osaan. Yksitoista prosessialuetta ja niihin kuuluvaa peruskäytännettä sisältävät ISO/IEC 21827 (2008) -standardin mukaan kaikki keskeiset turvallisuuden suunnittelun osa-alueet (taulukko 1). Loput yksitoista prosessialuetta sisältävät projektinhallintaan ja organisaatioon liittyvät käytänteet, kuten laadunvarmistuksen, konfiguraationhallinnan ja projektin riskienhallinnan. Vaikka nämä ovat tärkeitä osia turvallisuuden suunnittelun kokonaisuutta, liittyvät kyseiset prosessialueet kuitenkin enemmän projektinhallintaan ja organisaation prosesseihin, eivätkä varsinaisesti turvallisuuden suunnitteluun. Tämän vuoksi mainitut prosessialueet rajataan tämän tutkielman ulkopuolelle.

Prosessialueet (taulukko 1) sisältävät kuvauksen ja prosessialueen tavoitteet sekä siihen kuuluvat peruskäytänteet. Esimerkiksi prosessialueen PA05 ”Arvioi haavoittuvuus” (engl. assess vulnerability) tavoitteena on ymmärrys järjestelmän haavoittuvuuksista määritetyssä ympäristössä (ISO/IEC 21827, 2008). Prosessialue sisältää seuraavat peruskäytänteet (ISO/IEC 21827, 2008, s.39):

- BP.05.01 Valitse menetelmät, tekniikat ja kriteerit, jolla haavoittuvuudet tunnistetaan ja luokitellaan
- BP.05.02 Tunnista järjestelmän haavoittuvuudet
- BP.05.03 Kerää tiedot haavoittuvuuksien ominaisuuksista
- BP.05.04 Arvioi järjestelmän haavoittuvuus ja yhdistä samankaltaiset haavoittuvuudet
- BP.05.05 Valvo haavoittuvuuksia ja niiden muutoksia

Peruskäytänteet puolestaan sisältävät kuvauksen lisäksi esimerkit tuotoksista (engl. work products), joita peruskäytänteessä tehdään. Esimerkiksi peruskäytänne BP.05.02 Tunnista järjestelmän haavoittuvuudet (engl. identify security vulnerabilities) sisältää kaksi esimerkkituotosta, haavoittuvuuslistan (engl. vulnerability list) ja tunkeutumisprofiilin (engl. penetration profile). Ensin mainittu kuvaa järjestelmän haavoittuvuudet eri hyökkäyksille, ja jälkimmäinen sisältää haavoittuvuustestauksen tulokset. (ISO/IEC 21827, 2008.)

Tarkasteltaessa kyvykkyyden ulottuvuutta syvällisemmin huomataan, että siinä olevat yleiset käytänteet on jaettu edelleen loogisiin alueisiin, joita kutsutaan *yleisiksi piirteiksi* (engl. common features). Yleiset piirteet on puolestaan luokiteltu viiteen eri turvallisuuden kypsyystasoon (engl. capability levels), joiden tarkoituksena on arvioida organisaation kykyä suorittaa turvallisuuden suunnittelun prosesseja. (ISO/IEC 21827, 2008.) Esimerkiksi kyvykkyyden taso 2 sisältää seuraavat yleiset piirteet (ISO/IEC 21827, 2008, s.126):

- yleinen piirre 2.1 – toiminta on suunniteltua
- yleinen piirre 2.2 – toiminta on kurinalaista
- yleinen piirre 2.3 – toiminta on todennettua
- yleinen piirre 2.4 – toiminta on jäljitettyä

Yleinen piirre 2.1 sisältää kuvauksen lisäksi listan siihen liittyvistä yleisistä käytänteistä, jotka ovat seuraavat (ISO/IEC 21827, 2008, s.126):

- GP 2.1.1 – kohdenna resurssit
- GP 2.1.2 – määritä vastuut
- GP 2.1.3 – dokumentoi prosessi
- GP 2.1.4 – tarjoa työkalut
- GP 2.1.5 – varmista koulutus
- GP 2.1.6 – suunnittele prosessi

Yleiset käytänteet puolestaan sisältävät sanallisen kuvauksen lisäksi suhteet niihin liittyviin peruskäytänteisiin, kuten on esitetty aiemmassa esimerkissä (kuvio 1).

TAULUKKO 1 SSE-CMM prosessialueet (tiedot perustuvat lähteeseen ISO/IEC 21827, 2008)

Prosessialue	**Peruskäytänteiden-määrä**	**Tavoite**
PA01 Hallitse turvallisuustoiminnot (engl. Administer security controls)	4	Suunnitellut turvallisuustoiminnot ovat oikein konfiguroitu ja käytetty
PA02 Arvioi vaikutukset (engl. Assess impact)	6	Riskien vaikutukset järjestelmälle on tunnistettu ja kuvattu
PA03 Arvioi turvallisuusriski (engl. Assess security risk)	6	Turvallisuusriskit on tunnistettu ja priorisoitu
PA04 Arvioi uhka (engl. Assess threat)	6	Uhkat järjestelmän turvallisuudelle on tunnistettu ja kuvattu
PA05 Arvioi haavoittuvuus (engl. Assess vulnerability)	5	Järjestelmän haavoittuvuudet on tunnistettu määritetyssä ympäristössä
PA06 Rakenna luottamus turvallisuuteen (engl. Build assurance argument)	6	Tuotteet ja prosessit osoittavat selkeästi asiakkaan turvallisuustarpeiden huomioimisen
PA07 Koordinoi turvallisuutta (engl. Coordinate security)	4	Kaikki projektitiimin jäsenet ovat tietoisia tarvittavista turvallisuuden suunnittelun tehtävistä
PA08 Valvo turvallisuuden tilaa (engl. Monitor security posture)	7	Sisäiset ja ulkoiset turvallisuustapahtumat on tunnistettu ja jäljitetty
PA09 Tarjoa turvallisuuden tuotteet (engl. Provide security input)	6	Projektitiimin jäsenille tarjotaan kaikki tarvittava turvallisuuden informaatio kuten turvallisuusarkkitehtuuri tai suunnitelma
PA10 Määrittele turvallisuustarpeet (engl. Specify security needs)	7	Yleinen ymmärrys turvallisuuden tarpeista on saavutettu kaikilla osapuolilla
PA11 Verifioi ja validoi turvallisuus (engl. Verify and validate security)	5	Tehdyt ratkaisut kattavat turvallisuusvaatimukset

Näin peruskäytänteet ja osana niiden toteuttamista suoritettavat yleiset käytänteet muodostavat kokonaisuuden, jossa molemmat SSE-CMM:n ulottuvuudet, turvallisuuden suunnittelun käytänteet ja johtamistoiminta, on huomioitu.

SSE-CMM:n soveltuvuutta ohjelmistokehitykseen on arvioitu eri lähteissä, ja mallin osia on hyödynnetty uusien menetelmien kehittämisessä. Chanin ja Kwokin (2001) mukaan SSE-CMM on konkreettinen viitekehys turvallisten tietojärjestelmien suunnitteluun ja kehittämiseen, mutta se määrittelee vain sen, *mitä* tulee kehittää, jättäen avoimeksi *miten* tulee kehittää. Tämä on luonnollista, sillä SSE-CMM ei tarkoituksella esitä turvallisuuden suunnittelun prosessia, eikä ole näin ollen varsinaisesti turvallisuuden suunnittelun menetelmä tai prosessimalli. Chan ja Kwok (2001) esittävät turvallisen ohjelmistokehityksen prosessimallin (engl. Software Development Process for Secured Systems, SDPSS), joka perustuu SSE-CMM:n lisäksi yleisesti ohjelmistokehityksessä käytettyyn UML-mallintamiseen (Rumbaugh, Jacobson, & Booch, 1999). Malli on suunnattu erityisesti elektronisen kaupankäynnin järjestelmiin, ja siinä käytetään SSE-CMM-mallia viitekehyksenä määrittelemään tarkemmin elektronisen kaupankäynnin järjestelmien suunnittelussa ja kehittämisessä tarvittavia aktiviteetteja.

Myös Mellado, Fernández-Medina ja Piattini (2007) mainitsevat turvallisuuden suunnittelun standardien puutteeksi menetelmällisen tuen ja esittävät vastaavasti oman prosessimallin erityisesti turvallisuuden vaatimusmäärittelyyn. Melladon ym. (2007) prosessimallissa SSE-CMM:ää käytetään muun muassa turvallisuusvaatimusten arviointiin, mutta itse prosessimalli pohjautuu erityisesti Common Criterian (CC, 2012) käyttöön. Melladon ym. (2007) prosessimalli on esitelty tarkemmin seuraavassa alaluvussa.

Menetelmällisen tuen puute on mainittu myös muissa lähteissä. Lee, Lee ja Leen (2002) mukaan SSE-CMM ei erityisesti keskity turvallisten tietojärjestelmien kehittämiseen, mutta tarjoaa kuitenkin korkean tason viitekehyksen organisaation turvallisuustason määrittämiseen. Vastaavasti Siposen (2005) mukaan SSE-CMM ei suoraan sovellu tietojärjestelmien kehittämiseen.

Davis (2005) puolestaan tutkii olemassa olevia turvallisuuden menetelmiä ja standardeja, jotka tukevat tai voivat tukea turvallista ohjelmistokehitystä. Davisin (2005) mukaan turvallisuuden suunnittelussa on useita yleisesti hyväksyttyjä periaatteita, joihin SSE-CMM määrittelee kattavan viitekehyksen. Hänen mukaansa harva turvallisuuden suunnittelun menetelmä tukee ohjelmistokehityksen turvallisuutta alusta alkaen, johon SSE-CMM tekee poikkeuksen. Davis (2005) käsittelee kuitenkin tutkimuksessaan mallia melko suppeasti, eikä ota kantaa muun muassa siihen, että SSE-CMM ei määrittele käytettävää kehitysprosessia.

Kuten muun muassa Chanin ja Kwokin (2001), Melladon ym. (2007), Leen, Leen ja Leen (2002) sekä Siposen (2005) tutkimukset osoittavat, ei SSE-CMM ole suoraan sovellettavissa turvallisten ohjelmistojen tai tietojärjestelmien kehittämisen menetelmäksi. On kuitenkin huomioitavaa, että malli on tarkoitettu ensisijaisesti organisaation olemassa olevien prosessien turvallisuuden tason tarkastelun työkaluksi. SSE-CMM:ää voidaan niin ikään hyödyntää esitetyllä tavalla uusien turvallisten tietojärjestelmien kehittämismenetelmien luomisessa. Tässä tutkielmassa SSE-CMM:n on tarkoitus toimia esimerkkinä turvallisuuden suunnittelun standardista ja sen sovelluksista. Standardia käytetään myös viit-

teenä tarkasteltaessa turvallisuuden suunnittelun ja ketterän kehittämisen integrointia luvussa 4.

2.3 CC ja sen ohjelmistokehityksen sovelluksia

CC (engl. Common Criteria) (CC, 2012) on yleisesti käytetty informaatioteknologian tuotteiden turvallisuuden määrittelyn ja arvioinnin menetelmä, jonka kehitystyössä on ollut mukana useiden maiden turvallisuus-, standardointi- ja informaatioteknologian organisaatioita. CC on hyväksytty myös kansainväliseksi ISO 15408 –standardiksi (ISO/IEC 15408, 2008a; ISO/IEC 15408, 2008b). Menetelmä ei ota kantaa arvioitaviin tuotteisiin: sitä voidaan soveltaa laitteistoihin, laitteisto-ohjelmistoihin ja sovelluskehitykseen. Menetelmä on erittäin laaja, ja se on tarkoitettu niin käyttäjien, kehittäjien kuin turvallisuuden arvioijien menetelmäksi. Melladon, Fernández-Medinan ja Piattinin (2007) mukaan sen tarkoituksena on käyttäjien näkökulmasta määritellä turvallisuusvaatimukset, kehittäjien näkökulmasta määrittää turvallisuusominaisuudet tuotteelle ja arvioijien näkökulmasta tarkastella tuotteiden turvallisuusratkaisujen toimivuutta. Wäyrynen, Bodén ja Boström (2004) puolestaan kuvaavat CC-menetelmää yksinkertaisesti kieleksi, jolla voidaan ilmaista järjestelmän ja sen kehitysprosessin turvallisuusvaatimuksia.

Seuraavassa on esitetty muutamia menetelmän keskeisiä käsitteitä. Vaikka CC (2012) määrittelee käsitteet varsin laajasti, on alla kuvattu vain tämän tutkielman aihepiirin kannalta olennaisimmat käsitteet, ja niillä on tarkoitus täydentää yleiskuvaa menetelmästä. *Arvioinnin kohteella* (engl. Target Of Evaluation, TOE) tarkoitetaan joukkoa sovelluksia, laitteisto-ohjelmistoja ja/tai laitteistoja. Arvioinnin kohteen ei tarvitse välttämättä olla informaatioteknologian valmis tuote, vaan se voi olla osa siitä, joukko tuotteita tai teknologia, jota ei välttämättä koskaan tehdä tuotteeksi. Arvioinnin kohde voi olla myös yhdistelmä näistä. CC mainitsee esimerkkeinä arvioinnin kohteista muun muassa ohjelmistosovelluksen ja käyttöjärjestelmän tai niiden yhdistelmän, kortinlukijaan integroidun piirilevyn tai paikallisverkon laitteineen. *Suojausprofiili* (engl. Protection Profile, PP) tarkoittaa yleistä joukkoa turvallisuustarpeita tietylle arvioinnin kohteen (TOE) tyypille. Suojausprofiilit ovat riippumattomia toteutustavasta, joten tarvittavat mekanismit voidaan toteuttaa esimerkiksi millä tahansa ohjelmointikielellä tai teknologisella ratkaisulla. Suojausprofiilin tarkoituksena on määritellä korkealla tasolla tietyn tyyppinen teknologia (esimerkiksi palomuurit), eikä sen ole tarkoitus olla yksityiskohtainen kuvaus tietystä teknologiasta tai tuotteesta. *Turvallisuuskohde* (engl. Security Target, ST) voi liittyä yhteen tai useampaan suojausprofiiliin, ja sillä tarkoitetaan muun muassa kuvausta arviointikohteen turvallisuustavoitteiden täyttämisestä ja uhkien huomioon ottamisesta. Turvallisuuskohde sisältää myös esimerkiksi turvallisuusvaatimukset tietylle tuotteelle. (CC, 2012.)

Turvallisuuden vaatimusmäärittely on erittäin tärkeä osa turvallisten tietojärjestelmien kehittämistä (Mellado ym., 2007). CC sisältää kaksi turvallisuu-

den vaatimusten tyyppiä, jotka ovat turvallisuuden *varmistamisen vaatimukset* (engl. Security Assurance Requirements, SAR) ja turvallisuuden *toiminnalliset vaatimukset* (engl. Security Functional Requirements, SFR). Turvallisuuden varmistamisen vaatimukset ovat perustana luottamuksen rakentamiselle siitä, että turvallisuuden toimenpiteet ovat tehokkaita ja oikein toteutettuja (Mellado ym., 2007). Toiminnalliset vaatimukset määrittävät turvallisuuden tavoitteet arviointikohteelle, ja ne ovat teknologiariippumattomia, täsmällisiä kuvauksia kohteen toiminnasta (CC, 2012). CC (2012) määrittää toiminnallisten vaatimusten komponentit ja rakenteen. On kuitenkin huomioitavaa, että kuvatut toiminnalliset vaatimukset eivät ole tarkoitettu ratkaisuksi kaikkiin tietojärjestelmien turvallisuuden ongelmiin, vaan ne toimivat lähinnä kattavana turvallisuusvaatimusten listana luotettavien tuotteiden kehittämisessä (ISO/IEC 15408, 2008a).

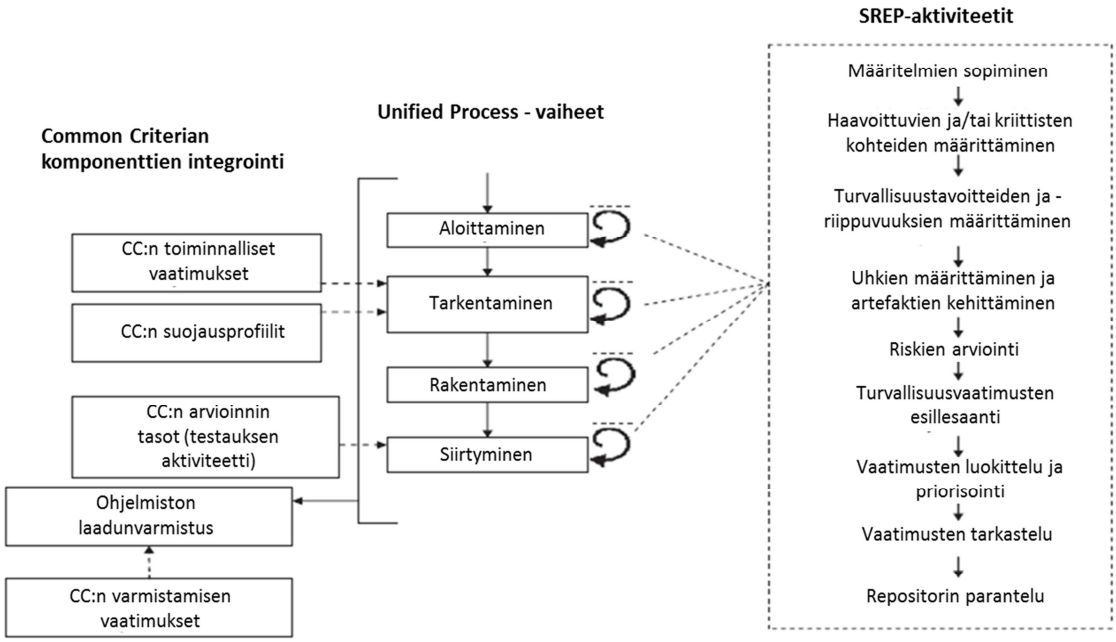
Malli sisältää turvallisuuden varmistamisen vaatimusten lisäksi seitsemän *arvioinnin tasoa* (engl. Evaluation Assurance Level, EAL), jotka kuvaavat hierarkkisesti tarkasteltavien kohteiden turvallisuuden tasoa (ISO/IEC 15408, 2008b). Tasot on esitetty taulukossa 2.

TAULUKKO 2 CC:n seitsemän turvallisuuden arvioinnin tasoa (tiedot perustuvat lähteeseen ISO/IEC 15408, 2008b)

Taso	**Kuvaus**
EAL1	Toiminnallisesti testattu
EAL2	Rakenteellisesti testattu
EAL3	Menetelmällisesti testattu ja tarkastettu
EAL4	Menetelmällisesti suunniteltu, testattu ja katselmoitu uudelleen
EAL5	Osittain formaalisti suunniteltu ja testattu
EAL6	Osittain formaalisti varmistettu suunnittelu ja testaus
EAL7	Formaalisti varmistettu suunnittelu ja testaus

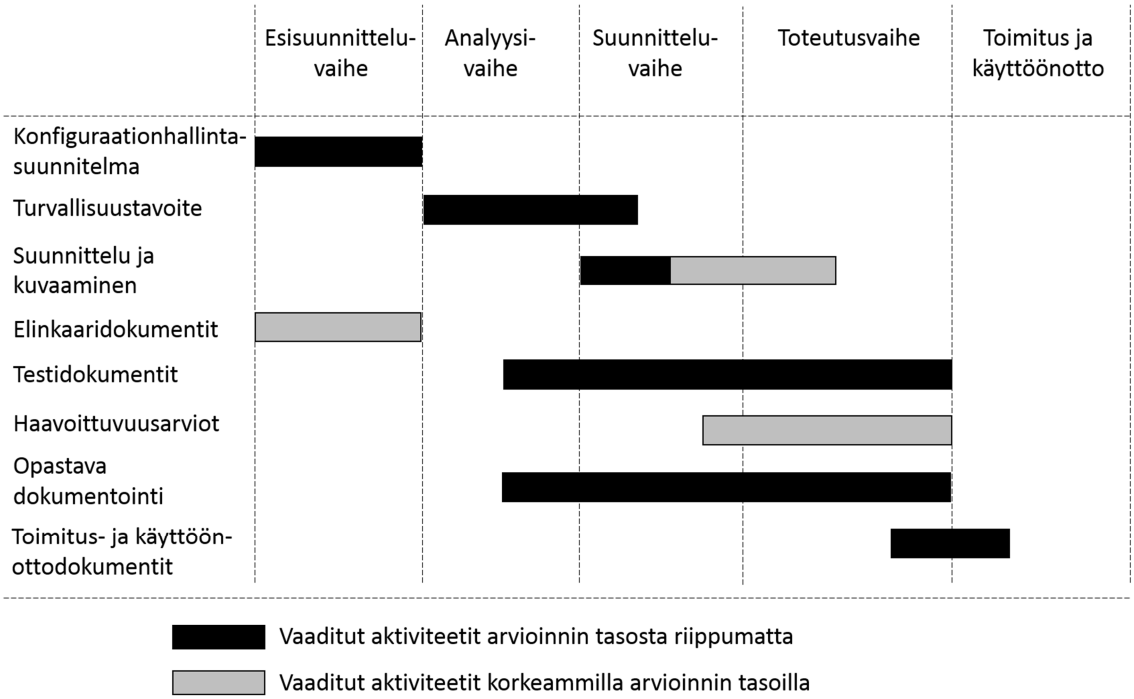
CC:n hyödyntämisestä turvallisten tietojärjestelmien kehittämisessä on olemassa joitakin aiempia tutkimuksia. Mellado ym. (2007) esittävät turvallisuuden vaatimusmäärittelyn prosessin (engl. Security Requirements Engineering Process, SREP), jossa turvallisuusvaatimukset integroidaan osaksi ohjelmistokehityksen prosessia (ks. kuvio 2). Prosessissa viitattu ohjelmistokehityksen menetelmä on UP (engl. Unified Process), joka on käyttötapaus- ja riskilähtöinen iteratiivinen kehitysmenetelmä (Jacobson, Booch & Rumbaugh, 1999). Turvallisuuden suunnitteluun on esitetty olemassa olevia turvallisuuden standardeja, joista prosessissa esitetyt aktiviteetit pohjautuvat erityisesti CC:n käyttöön. Melladon ym. (2007) prosessimallissa CC:n turvallisuuden toiminnalliset vaatimukset (SFR) on integroitu osaksi turvallisuuden vaatimusten määrittelyä. CC:n suojausprofiileja käytetään vakioituina kokonaisuuksina tietyntyyppisille turvallisuusvaatimuksille, jolloin samoja vaatimuksia voidaan soveltaa eri kohde-

alueissa. Mellado ym. (2007) mainitsevat esimerkkinä tällaisista vaatimuksista yksityisyyden suojan lainsäädännön. CC:n toiminnallisten vaatimusten ja suojausprofiilien lisäksi Melladon ym. (2007) mallissa käytetään CC:n turvallisuuden varmistamisen vaatimuksia (SAR) ja arvioinnin tasoja (EAL), joilla muun muassa varmistetaan turvallisuustavoitteiden saavuttaminen ja tunnistettujen uhkien huomioon ottaminen.



KUVIO 2 SREP turvallisuuden suunnittelun prosessimalli (Mellado ym., 2007, s.246)

Myös Vetterling, Wimmel ja Wisspeintner (2002) ovat soveltaneet osia CC:stä omassa tapaustutkimuksessaan ja prosessimallissa, jonka tarkoituksena on integroida turvallisuusnäkökohdat osaksi ohjelmistokehityksen prosessia (ks. kuvio 3). Vaikka Vetterlingin ym. (2002) prosessimallissa on esitetty perinteiset vaiheet, tapahtuu kehittäminen kuitenkin iteratiivisesti: yksi iteraatio sisältää suunnittelu-, analyysi-, mallinnus-, toteutus- ja toimitusvaiheet. *Esisuunnitteluvaiheessa* tehdään toteuttamisen ja sen vaatiman työn arvion lisäksi tekniset esivaatimukset ja korkean tason vaatimusmäärittely. CC:n yhteensopivuuden varmistamiseksi suunnitteluvaiheessa tulee tehdä konfiguraationhallinnan suunnitelma, jossa kuvataan konfiguraationhallinnan menetelmät ja työkalut. Korkeammilla EAL-tasoilla suunnitteluvaiheessa vaaditaan myös dokumentointi elinkaaren tuesta. *Analyysivaiheessa* tehdään vaatimusmäärittely ja turvallisuuskohteen (ST) määrittely. Turvallisuuskohteen määrittely jatkuu myös seuraavassa vaiheessa, koska CC:n mukaisesti toiminnallisissa turvallisuusvaatimuksissa vaaditaan yksityiskohtaisempaa mallintamista. Myös CC:n mukaisten ohjeistuksen dokumenttien ensimmäiset versiot laaditaan analyysivaiheessa, sisältäen muun muassa käyttäjä- ja järjestelmänvalvojan ohjeet.



KUVIO 3 Ohjelmistokehityksen ja CC:n integroitu prosessimalli (Vetterling ym., 2002, s. 131)

Analyysivaiheessa aloitetaan myös testauksen dokumentaation laatiminen. *Suunnitteluvaiheessa* määritellään järjestelmän arkkitehtuuri ja eri järjestelmäkomponentit. CC:n turvallisuuden varmistamisen vaatimukset, kuten toiminnallinen kuvaus tai järjestelmän suunnitelma kuvataan tässä vaiheessa, mutta dokumenttien laajuus riippuu vaaditusta EAL-tasosta. Myös haavoittuvuuden arviointi aloitetaan tässä vaiheessa korkeammilla EAL-tasoilla. *Toteutusvaiheessa* päätehtävänä on itse järjestelmän toteutus ja testaus, joka sisältää CC:n vaatimat testien analyysit ja dokumentit. Korkeammilla EAL-tasoilla vaaditaan myös toteutuksen mukainen suunnitteludokumentaatio ja haavoittuvuuden arvio. Tuotteen toimittamiseen liittyvien dokumenttien laatiminen aloitetaan jo tässä vaiheessa, ennen niiden käyttöönottoa. Toimitus- ja käyttöönottodokumentit viimeistellään *toimitus- ja käyttöönottovaiheessa*, jossa tuote otetaan operatiiviseen käyttöön. (Vetterling ym., 2002.)

CC:n avulla voidaan määrittää varsin kattavasti tietojärjestelmän turvallisuuden vaatimukset itse järjestelmälle ja sen kehitysprosessille (Mellado ym., 2007). Tietojärjestelmien kehittämisen kannalta CC:ssä esitetyt turvallisuuden tasot toimivat hyvänä arviointityökaluna esimerkiksi käytettävän kehitysmenetelmän turvallisuuden suunnittelun tasolle. CC ei kuitenkaan esitä itse menetelmää tai prosessia tietojärjestelmien kehittämiseen, joten malli on räätälöitävä tähän tarkoitukseen esimerkiksi Vetterlingin ym. (2002) tai Melladon ym. (2007) esityksen mukaisesti. Näin mahdollistetaan kehitystyön kannalta keskeisten osien hyödyntäminen. CC on erittäin laaja ja korkeampien arviointitasojen saavuttamiseksi tulee tuottaa paljon turvallisuuteen liittyviä määrittelyjä, malleja ja erilaisia kuvauksia, mikä voi asettaa haasteita tietojärjestelmien kehittämisen

kannalta. Tämän toteavat myös McDermot ja Fox (1999), joiden mukaan CC:n tuotteet ja niiden väliset riippuvuudet voivat olla vaikeita ymmärtää jopa vahvan teknisen taustan omaavalle henkilölle. Vaikka malli on sellaisenaan raskas tietojärjestelmien kehittämiseen, sen osia voidaan kuitenkin hyödyntää osana turvallista järjestelmäkehitystä muun muassa esitettyjen räätälöityjen menetelmien avulla.

2.4 Väärinkäyttötapaus

Käyttötapaukset (engl. Use Case) on yleisesti hyväksytty, yksinkertainen ja helposti omaksuttava olio-ohjelmoinnin käytänne, joka on laajasti käytössä järjestelmien vaatimusmäärittelyssä (McDermot & Fox, 1999; Kulak & Guiney, 2004). Käyttötapaukset mallintavat järjestelmän toiminnallisuutta käyttäjien, eli *aktorien* (engl. actors), näkökulmasta järjestelmän ja käyttäjän vuorovaikutuksena (Rumbaugh, Jacobson, & Booch, 1999). Aktori voi olla luonteeltaan ihminen (engl. human), esimerkiksi järjestelmän käyttäjä, tai tekninen (engl. non-human) kuten kirjanpitojärjestelmä (McDermot & Fox, 1999).

Negatiivisia skenaarioita on pitkään käytetty esimerkiksi sotilaallisten ja kaupallisten operaatioiden suunnittelussa (Alexander, 2002). McDermot ja Fox (1999) ovat esittäneet käyttötapauksista turvallisuuden suunnitteluun räätälöidyn version, *väärinkäyttötapaukset* (engl. abuse case). Väärinkäyttötapaukset määrittelevät käyttötapauksen tavoin vuorovaikutuksen aktorien ja järjestelmän välillä, mutta vuorovaikutuksen lopputulos on haitallinen joko itse järjestelmälle, jollekin sen aktoreista tai jollekin järjestelmän sidosryhmään kuuluvalle (McDermot & Fox, 1999). Väärinkäyttötapaukset on tarkoitettu turvallisuusvaatimusten –ja aktiviteettien ymmärtämisen tueksi, ja käytänne voidaan käsittää yhdeksi mallintamisen kieleksi (Siponen, 2005).

Väärinkäyttötapausten tulee olla täydellisiä, ja niiden tulee määrittää täsmällisesti tapauksesta aiheutuva haitta järjestelmälle. Käyttötapauksilla tulee myös osoittaa väärinkäyttöön tarvittava minimitaso järjestelmän käyttöoikeuksien suhteen; vaikka mikä tahansa haitta voidaan toteuttaa esimerkiksi täydellisellä järjestelmän kontrollilla, on tarkoituksenmukaisempaa kuvata haittaan tarvittava minimitaso. (McDermot & Fox, 1999.)

Väärinkäyttötapauksien aktorit ovat luonteeltaan samanlaisia ulkoisia agentteja kuin käyttötapauksissakin, mutta ne eivät kuitenkaan ole samoja aktoreita. Väärinkäyttötapaukset kuvataan siis erillisenä tavallisista käyttötapauksista, vaikka haitallinen aktori olisi sama molemmissa malleissa. Aktorit kuvataan tavallista käyttötapausta tarkemmin, sillä kuvaukset ovat hyödyllisiä väärinkäyttötapausten mallintamisessa. Keskeisiä piirteitä aktoreiden kuvaamisessa ovat aktorin resurssit, taidot ja tavoitteet. Resursseilla tarkoitetaan käyttötapaukseen tarvittavia työkaluja, henkilöitä tai organisaatioita sekä aikaa. Taidoilla kuvataan aktorin tekninen osaamistaso, ja tavoitteilla puolestaan tarkoitetaan pitkän tähtäimen tavoitteita, kuten esimerkiksi teollisuusvakoilua tai terrorismia. (McDermot & Fox, 1999.)

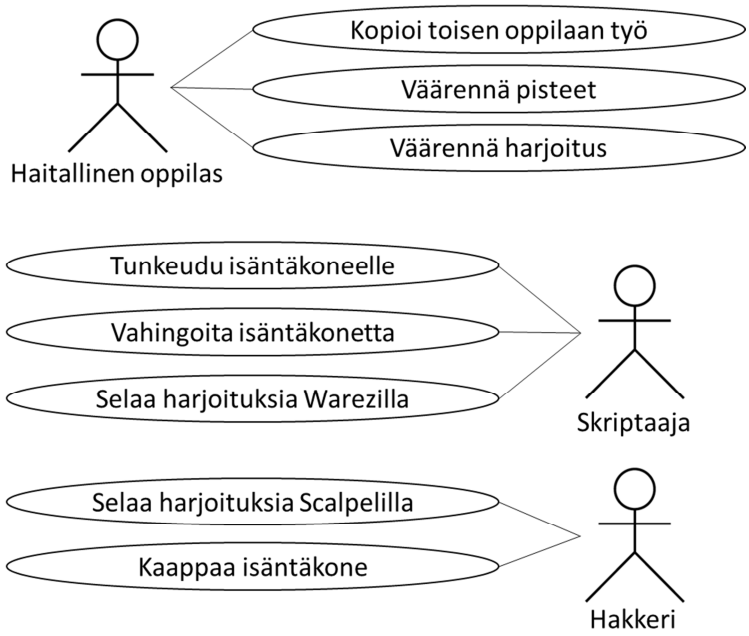
Väärinkäyttötapausten kuvaukset ovat niin ikään vastaavia kuin tavallisissa käyttötapauksissa, ja ne kuvataan tekstimuotoisena kertomuksena. Kuvaukset voivat kuitenkin erota tietyissä tilanteissa tavallisista käyttötapauksista; normaalisti käyttötapaukset keskittyvät yhteen vuorovaikutustilanteeseen, kun taas väärinkäyttötapauksissa voidaan kuvata kokoelma haitallisia vuorovaikutustilanteita. Tämä johtuu siitä, että turvallisuuspuutteita sisältävästä järjestelmän osasta ei voida olla varma; lopullinen väärinkäyttötapaus voi käyttää hyväksi vaatimusmäärittelyn erehdyksiä, suunnitteluvirheitä tai virheitä toteutuksessa. Koska väärinkäyttötapausten tarkoituksena on vähentää näiden ongelmien vaikutuksia, väärinkäyttötapaus voi kuvata monta abstraktia vuorovaikutustilannetta, joiden tavoitteena on lopulta sama väärinkäyttötapaus. Käyttötapausten kuvaamisessa käytetään apuna puurakennetta, jossa juuri on mallinnettava järjestelmä ja lehdet resursseja tai komponentteja väärinkäyttötapauksen kohteesta. Sisäiset solmukohdat ovat alijärjestelmiä, sovelluksia tai yksittäisiä luokkia ohjelman sisällä, jolloin jokainen polku juuresta lehtiin osoittaa, mitä järjestelmän osia voidaan käyttää väärin haitan tuottamisessa. (McDermot & Fox, 1999.)

Väärinkäyttötapaukset mallinnetaan hieman myöhemmin kuin tavalliset käyttötapaukset, sillä väärinkäyttötapauksien rakentamisessa käytetään käyttötapausten vastaavia järjestelmän osia. Väärinkäyttötapaukset luodaan seuraavan prosessin mukaisesti (McDermot & Fox, 1999, s.6):

1. Nimeä aktorit
2. Nimeä väärinkäytön tapaukset
3. Kuvaa väärinkäytön tapaukset
4. Tarkasta epäkohdat
5. Tarkasta yhtenäisyys ja minimaalisuus

Aktoreiden nimeäminen tehdään sen jälkeen, kun tavallisten käyttötapausten aktorit on määritetty, sillä käyttötapauksen ja väärinkäyttötapauksen aktori voi olla myös sama. Vaatimusmäärittelyn dokumentit voivat olla hyödyllisiä aktoreiden tunnistamisessa, mutta osana vaihetta tulisi tehdä järjestelmän ympäristön huolellinen analyysi. Aktoreiden nimeämisen jälkeen määritellään jokaiselle aktorille niiden vuorovaikutus järjestelmään ja nimetään väärinkäyttötapaukset. Tässä vaiheessa käytetään apuna UML-notaatioon (Rumbaugh ym., 1999) perustuvia väärinkäytön diagrammeja, jotka ovat samanlaisia kuin tavallisissa käyttötapauksissa.

Kuviossa 4 on esimerkki väärinkäyttötapauskaaviosta, joka sisältää kahdeksan väärinkäyttötapausta (ellipsit) ja kolme aktoria (henkilö-symbolit). Väärinkäyttötapaukset kuvataan, kun järjestelmän rajapinnat ja komponentit on riittävällä tasolla määritetty. Kuvauksia voidaan täydentää samalla, kun järjestelmän kuvausta täydennetään. Lopuksi tarkastetaan epäkohdat (engl. granularities) muun muassa liian suuren väärinkäyttötapausten määrän tai puutteellisuuksien suhteen ja varmistetaan, että jokainen käyttötapaus kuvaa haitallisen vuorovaikutuksen minimitasoa järjestelmään. (McDermot & Fox, 1999.)



KUVIO 4 Esimerkki väärinkäyttötapauskaaviosta (McDermot & Fox, 1999, s. 7)

Väärinkäyttötapauksista on esitetty McDermotin ja Foxin (1999) lisäksi myös muita vastaavia käytänteitä. Sindren ja Opdahlin (2005) malli esittää väärinkäyttötapaukset erillisen mallintamisen sijaan suhteessa muihin järjestelmän käyttötapauksiin, jolloin tavallisten käyttötapausten ja väärinkäyttötapausten yhteydet voidaan kuvata paremmin. Myös Aleksander (2002) käsittelee tapaustutkimuksessaan väärinkäyttötapauksia samassa yhteydessä tavallisten käyttötapausten kanssa. Vaikka McDermotin ja Foxin (1999) menetelmässä käyttötapaukset ovatkin erillisiä, tapahtuu niiden määrittely kuitenkin muiden käyttötapausten määrittelyn rinnalla. Näin turvallisuuden suunnittelu ja muu järjestelmän mallintaminen eivät ole täysin erillisiä prosesseja.

Firesmithin (2003) mukaan väärinkäyttötapaukset ovat tehokkaita turvallisuusuhkien analysointiin, koska niiden onnistumisen kriteerinä on onnistunut hyökkäys järjestelmää vastaan. Hänen mukaansa ne eivät kuitenkaan sovellu turvallisuusvaatimusten analysointiin ja määrittämiseen, johon Firesmith (2003) esittää turvallisuusvaatimuksiin keskittyvää versiota käyttötapauksista. Vaikka esitystä ei voida suoraan verrata McDermotin ja Foxin (1999) esitykseen, ei turvallisuusvaatimuksia ole jätetty täysin huomiotta väärinkäyttötapauksissa. McDermotin ja Foxin (1999) mukaan väärinkäyttötapauksilla voidaan lisätä asiakkaan ymmärrystä tuotteen turvallisuusominaisuuksista vaatimusmäärittelyn yhteydessä, koska väärinkäyttötapausten avulla voidaan määrittää sovelluksen toimintaympäristön uhkat ja sovelluksessa tarvittavat turvallisuusominaisuudet.

Turvallisuuden suunnittelu tulee ottaa turvallisten tietojärjestelmien kehittämisessä huomioon koko kehitysprosessin ajan (Mouratidis ym., 2003). Toisin kuin laajat ja formaalit turvallisuuden suunnittelun menetelmät ja -standardit,

kuten CC (CC, 2012) ja SSE-CMM (ISO/IEC 21827, 2008), McDermotin ja Foxin (1999) lähestymistapa on helppo omaksua sellaisenaan osaksi tietojärjestelmien kehittämistä muun muassa yhteisen notaation vuoksi. Myös Siposen (2005) mukaan käytänne on yhteensopiva tietojärjestelmien kehittämisen menetelmiin, jotka hyödyntävät mallintamisessa käyttötapauksia. Käytänne on yksinkertainen ja kehitetty selkeästi ohjelmistokehityksen näkökulmasta, eikä se näin ollen ole kaiken kattava turvallisuuden suunnittelun malli. Tämän toteavat myös McDermot ja Fox (1999) itse, ja heidän mukaansa väärinkäyttötapaukset toimivat hyödyllisenä turvallisuuden suunnittelun apuvälineenä eri ohjelmistokehityksen vaiheissa. Jos turvallisen ohjelmistokehityksen kohteena on korkeaa turvallisuustasoa vaativa ympäristö, voivat väärinkäyttötapaukset siten vaatia esimerkiksi erillisen turvallisuuden suunnittelun prosessin tai turvallisen ohjelmistokehityksen menetelmän, jossa väärinkäyttötapauksia käytetään täydentävänä käytänteenä osana turvallisuuden suunnittelun laajempaa kokonaisuutta. Esimerkkinä tällaisesta menetelmästä on Bacan ja Carlssonin (2011) turvallisuuden aktiviteeteilla laajennettu ketterän kehittämisen menetelmä, jota tarkastellaan kohdassa 4.3.3.

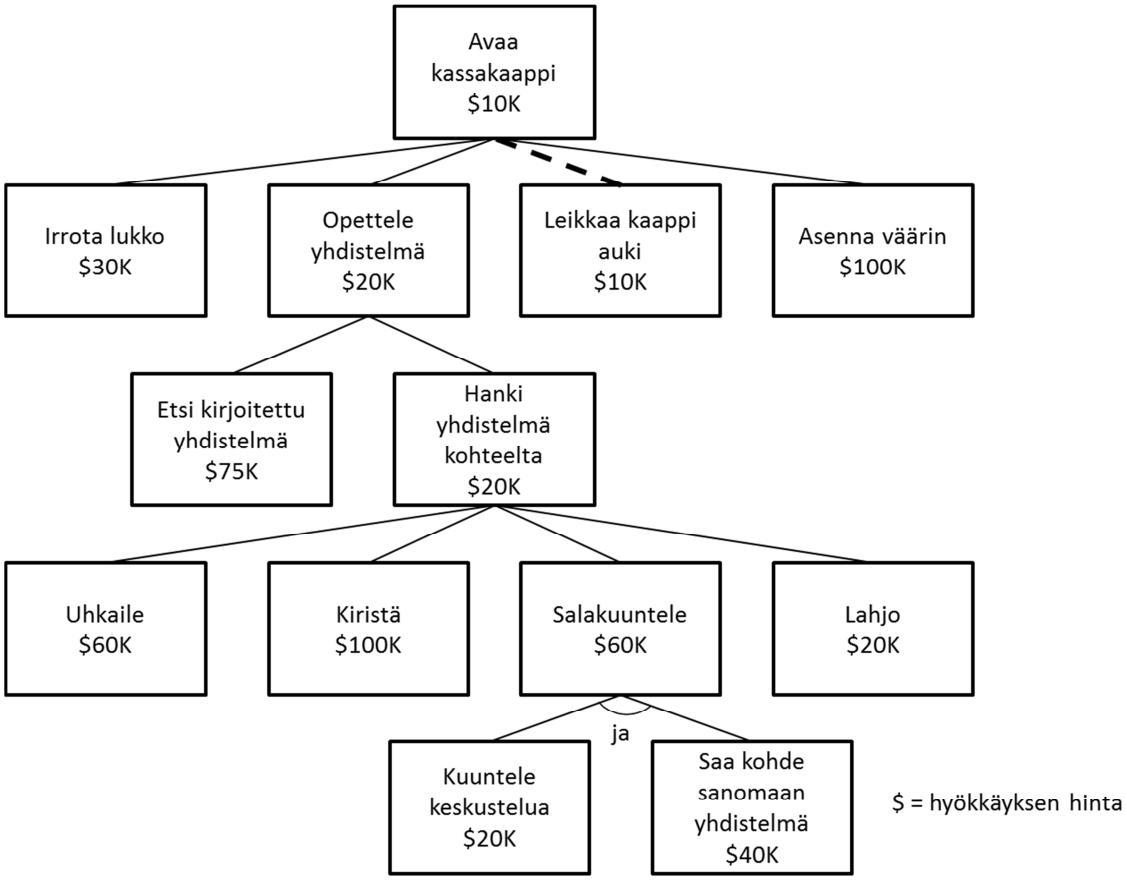
2.5 Hyökkäyspuut

Hyökkäyspuut (engl. attack trees) (Schneier, 1999) ovat käytänne, jolla voidaan kuvata järjestelmien ja alijärjestelmien turvallisuutta. Hyökkäyspuut mallintavat hyökkäysten vaikutuksia järjestelmään ja auttavat tätä kautta järjestelmän turvallisuuden parantamisessa (Schneier, 1999). Järjestelmään kohdistuvat hyökkäykset voidaan luokitella hyökkäyspuiden avulla järjestelmällisesti, ja graafinen puurakenne on helppo omaksua (Mauw & Oostdijk, 2006). Seuraavaksi esitellään hyökkäyspuiden perusperiaatteet.

Hyökkäyspuun *juurisolmu* (engl. root node) on hyökkäyksen päämäärä, ja *lehtisolmut* (engl. leaf nodes) ovat varsinaisia hyökkäyksiä. Järjestelmässä voi olla useita juurisolmuja, jolloin jokainen niistä mallintaa eri päämääriä ja on siten erillinen hyökkäyspuu. Hyökkäyspuussa on ”tai”- solmuja (engl. ”or” nodes), jotka kuvaavat eri tapoja saman tavoitteen (juurisolmun) saavuttamiseen. Näiden lisäksi hyökkäyspuuhun kuuluu ”ja” - solmuja (engl. ”and” nodes), jotka puolestaan kuvaavat tavoitteen saavuttamiseksi vaadittavat vaiheet. Kun näistä on muodostettu hyökkäyspuu, voidaan lehtisolmuille määrittää eri arvoja, joiden perusteella hyökkäyspuun oksia voidaan karsia. Näitä arvoja voivat olla esimerkiksi totuusarvot, rahallinen arvo, todennäköisyysarvo tai jokin näiden yhdistelmä. Solmukohdan arvo on sen alla olevien solmujen funktio, ja arvo lasketaan lehdestä juureen päin. (Schneier, 1999.)

Kuviossa 5 esitetään esimerkki yksinkertaisesta hyökkäyspuusta, joka mallintaa kassakaapin avaamisen edullisinta hyökkäystä. Hyökkäyksen päämääränä on kassakaapin avaaminen (juurisolmu), ja sen alla olevat lehtisolmut kuvaavat mahdollisia hyökkäyksiä. Jokaiselle lehtisolmulle on asetettu arvo, joka on esimerkissä hyökkäyksen rahallinen arvo. Eri laskentasäännöt päte-

vät ”ja-”, sekä ”tai”-solmuille, jolloin esimerkiksi salakuuntelussa on laskettava mukaan sekä keskustelun kuuntelun että yhdistelmän saamisen kustannukset. Tai-solmuissa puolestaan lasketaan mukaan edullisin vaihtoehto, joka tuottaa päämäärän saavuttamisen edullisimmaksi vaihtoehdoksi kassakaapin avaamisen leikkaamalla (katkoviiva kuviossa 5). Notaatio voi olla kuvion 5 kaltainen puurakenne, jossa ”ja”-solmut erotetaan ”tai”-solmuista yhdistävällä viivalla, mutta puurakenne voidaan kuvata myös tekstiluettelona, jolloin eri tasot kuvataan hierarkkisesti numeroilla ja tekstin perään kirjoitetaan (JA) tai (TAI). (Scneier, 1999.)



KUVIO 5 Esimerkki hyökkäyspuusta (Schneier, 1999, s.19)

Vaikka esimerkki on yksinkertaistettu eikä rahallisen arvon käyttäminen ole välttämättä uhkien mallintamisessa järkevää, kuvaa se kuitenkin hyvin hyökkäyspuiden muodostamisen periaatteen. Vastaavalla tavalla voidaan mallintaa hyökkäyspuu esimerkiksi turvallisuuskriittisen ohjelmistokomponentin saastuttamiselle käyttäen eri hyökkäysvektoreiden todennäköisyyksiä. Lopputuloksena saadaan malli, jonka avulla voidaan arvioida toteutettavien vastatoimien tarpeellisuutta sekä määrittää itse vastatoimet.

Hyökkäyspuut liitetään yleisesti Schneieriin (1999) (Mauw & Oostdijk, 2006), mutta esimerkkejä vastaavista käytänteistä löytyy muun muassa Amorosolta (1994) sekä Phillipsiltä ja Swileriltä (1998). Eri esityksillä on kuitenkin sama päämäärä, jossa hyökkäyspuulla arvioidaan järjestelmän haavoittuvuutta

eri hyökkäyksille. Puiden avulla voidaan määrittää esimerkiksi tietyn totuusarvon läsnäolo, erityyppiset haavoittuvuudet tai hyökkääjän kustannukset, jotka alittavat tietyn rajan (Schneier, 1999). Nämä päämäärät ovat yhteisiä eri käytänteille riippumatta sen esittäjästä. McDermotin ja Foxin (1999) väärinkäyttötapausten tapaan hyökkäyspuut on helppo omaksua yhtenä käytänteenä osaksi järjestelmäkehitystä, ja hyökkäyspuita hyödynnetään muun muassa kohdissa 4.3.2 ja 4.3.5 käsiteltävissä Singhalin ja Singhalin (2011) viitekehyksessä sekä Kongslin (2006) turvallisen ja ketterän Web-kehityksen prosessissa.

2.6 Yhteenveto

Tässä luvussa käsiteltiin tietojärjestelmien turvallisuuteen liittyviä käsitteitä, turvallisen tietojärjestelmän kehittämisstandardien ja –menetelmien yleisiä piirteitä, niiden käsitteitä sekä käytänteitä. Yleinen tietoturvastandardi ja turvallisuuden suunnittelun kypsyysmalli SSE-CMM (ISO/IEC 21827, 2008) sisältää turvallisuuden suunnittelun keskeiset peruskäytänteet ja niiden tuotokset. Sen avulla voidaan myös määrittää kattavasti organisaation turvallisuusprosessien kypsyystaso. SSE-CMM todettiin laajaksi viitekehykseksi, jota ei kuitenkaan voida sellaisenaan soveltaa turvallisessa ohjelmisto- ja järjestelmäkehityksessä sen menetelmällisen tuen puuttumisen vuoksi.

Toinen käsitelty tietoturvastandardi, Common Criteria (CC, 2012), on tarkoitettu informaatioteknologian tuotteiden turvallisuuden määrittelyyn ja arviointiin. Menetelmä on tuoteriippumaton, ja se sisältää muun muassa tuotteiden turvallisuuden varmistamisen vaatimukset ja turvallisuuden toiminnalliset vaatimukset. Tuotteiden turvallisuuden tasoa voidaan arvioida eri arviointitasoilla. CC todettiin erittäin laajaksi, mutta sellaisenaan raskaaksi järjestelmäkehitykseen. Menetelmää on kuitenkin sovellettu turvallisessa ohjelmistokehityksessä muun muassa aktiviteettien ja turvallisuusvaatimuksien osalta, joista käsiteltiin kaksi esimerkkiä.

Lopuksi esiteltiin käsitteet ja periaatteet kahdesta yksinkertaista mallintamisen käytänteestä, väärinkäyttötapauksista (McDermot & Fox, 1999) ja hyökkäyspuista (Schneier, 1999). Molemmat näistä todettiin helposti omaksuttavaksi ja yhteensopivaksi ohjelmistokehitykseen. Sellaisenaan kumpikaan ei kuitenkaan riitä yksittäisenä käytänteenä täyttämään turvallisuuden suunnittelun tarpeita, mutta ne ovat hyödyllisiä esimerkiksi osana turvallisuuden suunnittelun viitekehystä ja menetelmää.

3 KETTERÄ KEHITTÄMINEN

Tämän luvun tarkoituksena on tarkastella ketterää lähestymistapaa olemassa olevaan kirjallisuuteen ja Agile-manifestiin (Agile alliance, 2001) perustuen. Lisäksi esitellään prosessit, roolit ja käytänteitä kahdesta yleisestä ketterän kehittämisen menetelmästä, XP:stä ja Scrumista.

3.1 Agile-manifesti ja ketterän kehittämisen määritelmä

Ketterää ohjelmistokehitystä määriteltäessä viitataan usein Agile Allianssin (2001) julkaisemaan manifestiin, jonka ovat laatineet alan keskeiset vaikuttajat. Manifestin arvot kuvaavat hyvin ketterän kehittämisen ajatusmallia:

> 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
>
> Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän.
>
> (Agile Alliance, 2001.)

Arvojen lisäksi manifesti sisältää 12 periaatetta. Toimitettavaan ohjelmistoon liittyvät periaatteet korostavat aikaista ja jatkuvaa toimitusta, hyvää teknistä laatua ja suunnittelua sekä yksinkertaisuutta. Muuttuvat vaatimukset myös

myöhemmässä vaiheessa kehitystä ovat tervetulleita. Periaatteiden mukaisesti edistymisen mittarina on toimiva ohjelmisto. Kehittäjiin liittyvät periaatteet sisältävät projektien rakentamisen motivoituneiden henkilöiden ympärille, tiimien itseohjautuvuuden ja tehokkuuden sekä parempien työtapojen löytämisen. Kommunikaatio on myös tärkeää, ja paras tapa jakaa tietoa on kasvokkain tapahtuva keskustelu. Liiketoiminnan edustajien ja kehittäjien on työskenneltävä yhdessä päivittäin koko kehitystyön ajan. (Agile Alliance, 2001.)

Ketterän ohjelmistokehityksen määritelmä on esitetty lukuisissa teoksissa. Niistä on löydettävissä yleisesti Agile-manifestin arvot ja periaatteet. Abrahamsson, Salo, Ronkainen ja Warsta (2002) mainitsevat ketterän kehityksen piirteiksi inkrementaalisuuden, yhteistyöhakuisuuden, selkeyden ja mukautuvuuden. Ensin mainittu tarkoittaa pieniä ohjelmistojulkaisuja nopeissa sykleissä. Yhteistyöhakuisuudella tarkoitetaan asiakkaan ja kehittäjän tiivistä kommunikaatiota sekä jatkuvaa yhteistyötä. Selkeydellä tarkoitetaan itse menetelmän omaksumisen ja sen muokkaamisen helppoutta, kun taas mukautuvuudella viitataan joustavuuteen muutoksissa. (Abrahamsson ym., 2002.) Vaikka Agile-manifestin piirteet esiintyvät yleisesti ketterän kehittämisen määritelmissä, on manifesti Laantin (2013) mukaan paljon keskustelua aiheuttanut ja kiistelty dokumentti. Hänen mukaansa tämä johtuu osittain siitä, miten Agile-manifesti on ymmärretty tai eroavatko omat näkemykset sen sisällöstä. Laanti (2013) mainitsee, että ristiriitaisuuksien vuoksi manifestin selkeyttäminen on nähty tarpeelliseksi ja jopa osa ketterän kehittämisen vaikuttajista haluaisi tehdä päivityksiä sen sisältöön. Laanti (2013) viittaa ketterän kehittämisen asiantuntijatapaamiseen (Stevens, 2011), jossa 30 ketterän ohjelmistokehityksen vaikuttajaa kokoontui keskustelemaan manifestin sisällöstä. Keskustelun mukaan ketterän yhteisön tulisi:

- vaatia teknistä erinomaisuutta
- edistää yksilön muutosta ja johtaa organisaation muutosta
- koota tietämystä ja edistää koulutusta
- maksimoida arvon luomista koko prosessin läpi

Luettelosta puuttuu paljon alkuperäisen manifestin periaatteiden painopisteistä ja muut alueet, erityisesti teknisen erinomaisuuden vaatiminen, on nähty jopa alkuperäistä tärkeämmäksi. Näkökulma on siis muuttunut tiimi- ja yksilökeskeisyydestä järjestäytyneempään ja johdettuun suuntaan. (Laanti, 2013.) Väitettä tukee myös Boehm ja Turner (2003), joiden mukaan ohjelmistokehityksen trendi on menossa suuntaan, jossa tarvitaan paitsi ketteryyttä, myös kurinalaisuutta.

Ketterä ohjelmistokehitys on noussut suureen suosioon erityisesti 2000-luvulla (VersionOne, 2013; Rodriguez, Markkula, Oivo & Turula, 2012). Dybån ja Dingsøyrin (2008) mukaan ketterä kehittäminen edustaa merkittävää ohjelmistokehityksen muutosta perinteisistä, suunnittelulähtöisistä ja vaihejakoisista menetelmistä. Myös Cockburn ja Highsmith (2001) kirjoittavat ketterän kehittämisen näyttäneen hyödyllisyytensä perinteisiin menetelmiin verrattuna, ja

heidän mukaansa ketterä kehittäminen vastaa nykypäivän nopeiden markkinavoimien muutoksiin. Conboy (2009) täydentää näkemystä mainitsemalla ketterän kehitysmenetelmän piirteiksi jatkuvan valmiuden muutosten toteuttamiseen ja niiden ennakointiin, lisäten asiakkaalle arvoa taloudellisuuden, laadun ja yksinkertaisuuden näkökulmasta.

Muun muassa McCormickin (2001), Boehmin ja Turnerin (2003) sekä useiden muiden lähteiden mukaan ei ole olemassa yhtä ohjelmistokehityksen mallia, joka sopii kaikkiin tarkoituksiin. On selvää, että lukuisat eri tekijät kuten asiakasorganisaation toimiala, projektin monimutkaisuus ja tiimin koko sekä asiakkaan sitoutuminen vaikuttavat valitun kehitysmenetelmän toimivuuteen. Ketterä kehittäminen on Cockburnin ja Highsmithin (2001) mukaan parhaimmillaan ihmiskeskeisessä ja yhteistyöhakuisessa organisaatiossa. Tämä on luonnollista tarkasteltaessa ketterän kehityksen arvoja ja periaatteita, joissa korostetaan tiivistä yhteistyötä ja kommunikaation merkitystä. Boehmin ja Turnerin (2003) mukaan ketterällä lähestymistavalla voidaan hallita hyvin muutoksia ja näkymättömyyttä rakentamalla jaettu ymmärrys kehitettävästä järjestelmästä jokaisen tiimin jäsenen ”hiljaiseen tietoon”. Heidän mukaansa ketterä lähestymistapa ei kuitenkaan skaalaudu laajoihin ja monimutkaisiin projekteihin, eikä ketterä lähestymistapa tue kehitystyön kurinalaisuutta.

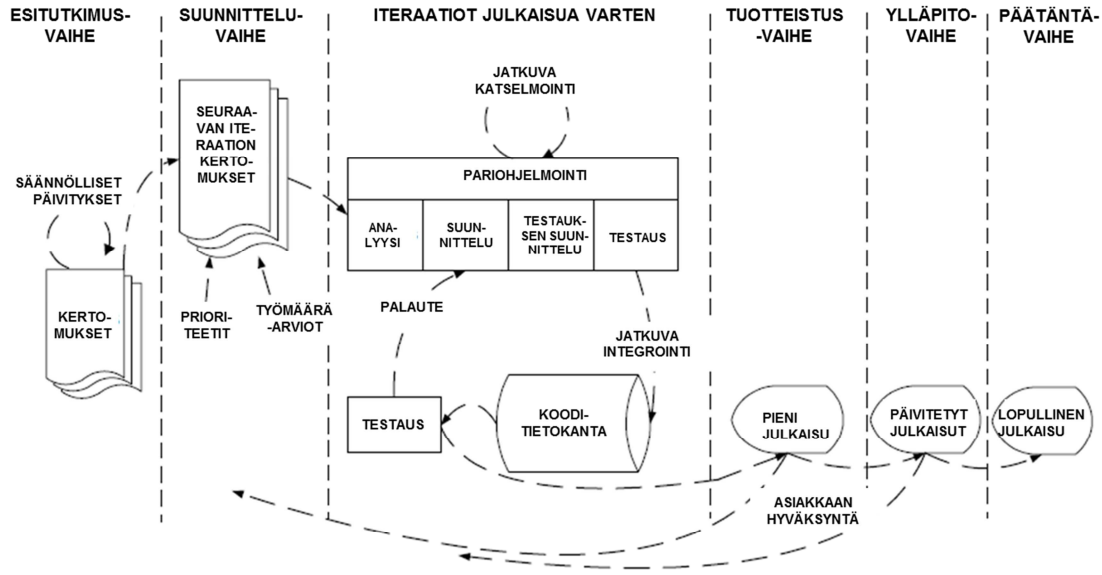
Ketterään ohjelmistokehitykseen on esitetty useita menetelmiä. Esimerkkejä ketterän kehityksen menetelmistä ovat Extreme Programming (XP) (Beck, 1999b; Beck & Andres, 2004), Scrum (Schwaber & Beedle, 2002 ; Schwaber & Sutherland 2013), Kanban (Anderson, 2010), Lean software development (Poppendieck & Poppendieck, 2003), Dynamic Systems Development Method (DSDM) (Stapleton, 1997), Crystal methods (Cockburn, 2002), Feature Driven Development (FDD) (Palmer & Felsing, 2001), Agile modeling (Ambler, 2002) ja Adaptive Software Development (ASD) (Highsmith, 2013). Seuraavissa alaluvuissa esitellään laajemmin kaksi ensin mainittua, koska ne ovat yleisimmin käytettyjä.

3.2 XP

XP (engl. eXtreme Programming) (Beck, 1999b; Beck & Andres, 2004) on suosittu ketterän ohjelmistokehityksen menetelmä, joka on saanut vaikutteita ohjelmistokehityksen hyviksi havaituista käytänteistä (Beck, 1999a). Kuten useat muut ketterät menetelmät, on XP syntynyt vastaamaan perinteisten menetelmien pitkien kehityssyklien ongelmiin ja menetelmä pyrkii onnistuneeseen ohjelmistokehitykseen muuttuvista tai epämääräisistä vaatimuksista huolimatta. Pitkälle tähtäävän suunnittelun ja analyysin sijaan XP:ssä tehdään perinteisen ohjelmistokehityksen aktiviteetteja iteratiivisesti koko kehitystyön ajan. (Beck, 1999a.)

Ohjelmistokehityksen elinkaari koostuu XP:ssä kuudesta vaiheesta (kuvio 6), jotka ovat esitutkimusvaihe (engl. exploration phase), suunnitteluvaihe (engl. planning phase), iteraatiot julkaisua varten (engl. iterations to release), tuotteis-

tusvaihe (engl. productionizing phase), ylläpitovaihe (engl. maintenance phase) ja päätäntävaihe (engl. death) (Abrahamsson ym., 2002, s.19).



KUVIO 6 XP:n prosessi (Abrahamsson ym., 2002, s.19)

Esitutkimusvaiheessa projektitiimi totuttelee projektissa käytettävään teknologiaan ja työkaluihin. Kehitettävästä järjestelmästä tehdään myös prototyyppi käytettävän teknologian ja arkkitehtuurivaihtoehtojen tutkimiseen. Asiakkaat kuvaavat tutkimusvaiheessa ensimmäiseen julkaisuun sisällytettävät toiminnot *kertomuksiksi*, jotka kirjoitetaan korteille (engl. story cards). Tutkimusvaihe kestää projektista riippuen muutamasta viikosta muutamiin kuukausiin. (Beck, 1999b; Abrahamsson ym., 2002.)

Suunnitteluvaiheessa asiakkaiden kertomukset priorisoidaan ja kehittäjät arvioivat niiden vaatiman työn. Muutaman päivän kestävä suunnitteluvaihe tuottaa ensimmäisen julkaisun sisällön ja sen aikataulun, joka on normaalisti alle kaksi kuukautta. (Beck, 1999b; Abrahamsson ym., 2002.)

Iteraatiot julkaisua varten sisältää suunnitteluvaiheen aikataulun ja sisällön pilkkomisen lukuisiin, yhdestä neljään viikkoa kestäviin iteraatioihin. Ensimmäisessä iteraatiossa kehitetään koko järjestelmän arkkitehtuuri valitsemalla kertomukset, jotka edellyttävät järjestelmän rakentamisen. Iteraation lopussa tehdään toiminnalliset testit. XP:ssä asiakas luo toiminnalliset testit ja valitsee myös kertomukset kuhunkin iteraatioon. (Beck, 1999b; Abrahamsson ym., 2002.)

Viimeistä iteraatiota seuraavassa *tuotteistusvaiheessa* tehdään järjestelmän testausta ja laadun varmistamista. Mahdollisten uusien vaatimusten osalta päätetään, lisätäänkö ne nykyiseen vai tulevaan julkaisuun. (Beck, 1999b; Abrahamsson ym., 2002.)

Ylläpitovaihe seuraa ensimmäisen julkaisun jälkeen, ja se sisältää järjestelmän tuotantokäytössä tarvittavat tehtävät kuten asiakastuen sekä mahdolliset uudet iteraatiot. (Beck, 1999b; Abrahamsson ym., 2002.)

Päätäntävaiheessa asiakkaalla ei ole enää uusia kertomuksia toteutettavaksi ja järjestelmä täyttää sille asetetut vaatimukset. Päätäntävaihe voi myös päättyä tilanteeseen, jossa järjestelmä ei täytä asetettuja vaatimuksia tai sen jatkokehitys nähdään liian kalliiksi. (Beck, 1999b; Abrahamsson ym., 2002.)

XP:ssä on määritelty prosessin lisäksi joukko rooleja ja niihin kuuluvia tehtäviä, jotka on esitetty alla (Beck, 1999b; Abrahamsson ym., 2002):

- *Ohjelmoija* (engl. programmer) pitää ohjelmakoodin mahdollisimman yksinkertaisena ja kirjoittaa testejä. Ohjelmoijan roolissa on tärkeää kommunikoida muiden ohjelmoijien ja tiimin jäsenten kanssa.
- *Asiakas* (engl. customer) kirjoittaa käyttäjäkertomuksien lisäksi toiminnalliset testit. Asiakas määrittää vaatimusten toteuttamiselle prioriteetit ja päättää milloin vaatimus on täytetty.
- *Testaaja* (engl. tester) tekee toiminnalliset testit ja auttaa asiakasta kirjoittamaan testejä. Testaaja myös ylläpitää testaukseen liittyviä työkaluja ja julkaisee testitulokset.
- *Seuraaja* (engl. tracker) valvoo jokaisen iteraation edistymistä ja arvioi asetettavia tavoitteita sekä antaa palautetta tiimille esimerkiksi työmääräarvioista tulevien arvioiden parantamiseksi.
- *Valmentaja* (engl. coach) on vastuussa koko kehitysprosessista. Valmentajan roolissa XP:n tunteminen on tärkeää, jotta voidaan ohjata muita tiimin jäseniä prosessin noudattamisessa.
- *Konsultti* (engl. consultant) auttaa tarvittaessa tiimiä ulkoisena jäsenenä esimerkiksi tietyn teknisen ongelman ratkaisussa.
- *Johtaja* tekee päätökset ja kommunikoi tiimin kanssa päätöksenteon tueksi.

Roolien lisäksi XP:ssä on esitetty seuraavat käytänteet (Beck, 1999a):

- *Suunnittelupelissä* (engl. planning game) ohjelmoijat arvioivat käyttäjien kertomusten toteutuksen vaatiman työmäärän, jonka jälkeen asiakas päättää julkaisujen sisällön ja ajoituksen.
- *Pienet julkaisut* (engl. small releases) tarkoittavat järjestelmän tuotteistusta muutaman kuukauden sisällä. Uusia julkaisuja tehdään kuukausittain tai jopa päivittäin.
- *Metaforat* (engl. metaphor) ovat järjestelmän kuvauksia, jotka auttavat kehittäjiä ja asiakasta ymmärtämään järjestelmän toimintaa.
- *Yksinkertainen suunnittelu* (engl. simple design) tarkoittaa järjestelmän toteuttamista mahdollisimman yksinkertaisesti ilman päällekkäisyyksiä tai turhaa monimutkaisuutta.

- *Testaus* (engl. testing) on asiakaslähtöistä; asiakkaat kirjoittavat käyttäjäkertomusten toiminnalliset testit, ja kehittäjät tekevät yksikkötestausta jatkuvasti. Testit täytyy läpäistä asianmukaisesti.
- *Refaktorointi* (engl. refactoring) tarkoittaa järjestelmän kehittämistä jatkuvasti olemassa oleviin suunnitelmiin perustuen.
- *Pariohjelmoinnissa* (engl. pair programming) kaksi henkilöä kirjoittaa koodia samalla työasemalla.
- *Yhteisomistajuudessa* (engl. collective ownership) kuka tahansa voi muuttaa mitä tahansa osaa toteutuksesta.
- *Jatkuva integrointi* (engl. continuous integration) tarkoittaa uuden koodin lisäämistä ja integrointia järjestelmään useita kertoja päivässä.
- *Asiakkaan läsnäolo* (engl. on-site customer) on käytänne, jossa asiakkaan on oltava jatkuvasti tiimin tavoitettavissa.
- *40-tunnin viikolla* (engl. 40-hour weeks) tarkoitetaan työviikon 40-tunnin maksimipituutta. Kahta ylityöviikkoa ei saa tulla peräkkäin.
- *Avoin työtila* (engl. open workspace) on suuri sermeillä erotettu huone, jossa tiimi työskentelee.
- *Vain säännöt* (engl. just rules) tarkoittaa tiimin sääntöjä, joita sitoudutaan noudattamaan. Sääntöjä voidaan kuitenkin milloin tahansa muuttaa, jos muutosten seurauksista ollaan yksimielisiä.

XP :stä on julkaistu myös uudempi versio (Beck & Andres, 2004), jossa on esitetty useita muita käytänteitä. Käytänteet on jaettu ensisijaisiin (engl. primary) ja täydentäviin (engl. corollary) käytänteisiin, joista ensin mainittuun kuuluvat yhdessä istuminen, kokonaisvaltainen tiimi, informatiivinen työtila, tarmokas työ, pariohjelmointi, tarinat, viikkosykli, vuosineljännessykli, löysä, kymmenen minuutin ohjelmakooste, jatkuva integrointi, testilähtöinen ohjelmointi ja inkrementaalinen suunnittelu. Täydentäviä käytänteitä ovat asiakkaan todellinen osallistuminen, inkrementaalinen käyttöönotto, tiimin jatkuvuus, kutistuvat tiimit, ongelman alkuperän analysointi, yhteinen ohjelmakoodi, yksi ohjelmakoodikanta, ohjelmakoodi ja testit, päivittäinen käyttöönotto, neuvoteltavan mittakaavan sopimus ja käyttöön perustuvat maksut. (Beck & Andres, 2004.)

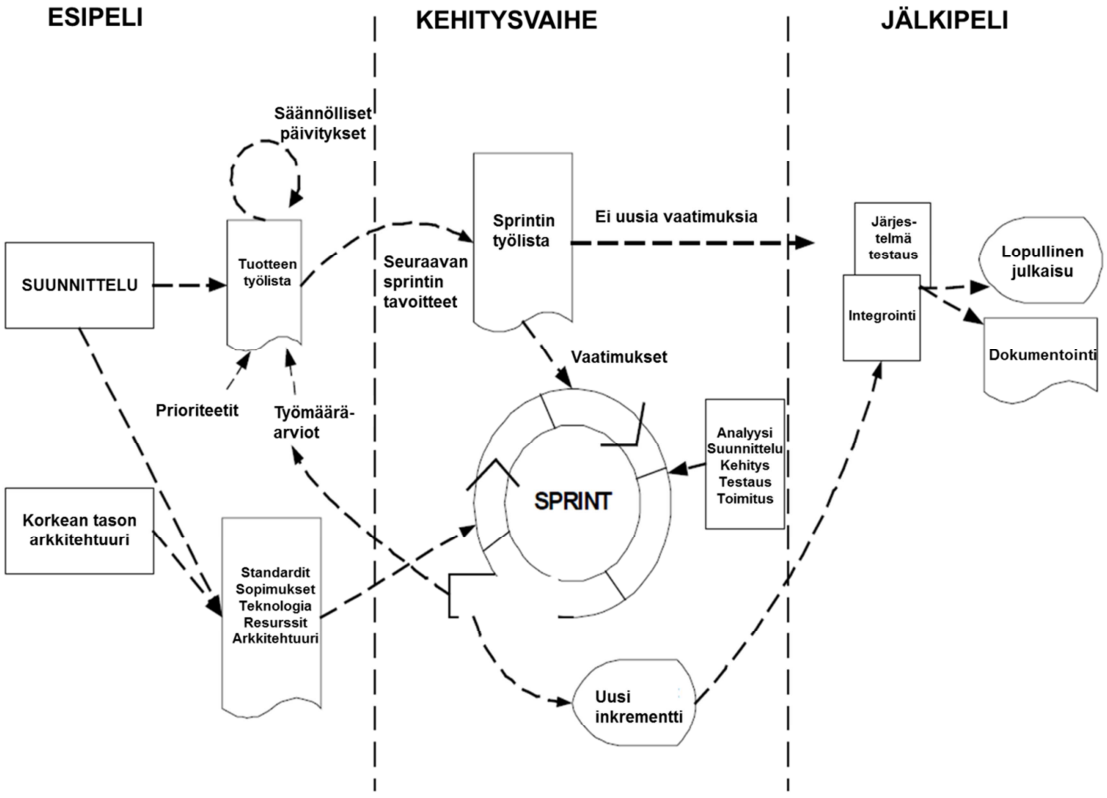
XP:n käytöstä on julkaistu useita tutkimuksia alan kirjallisuudessa. Dybån ja Dingsøyrin (2008) laajan empiirisen katsauksen mukaan XP:tä käyttävien organisaatioiden ohjelmistokehittäjät ovat tyytyväisiä paitsi työhönsä, myös itse tuotteeseen ja menetelmän käytänteet on helppo omaksua. Toisaalta pariohjelmointia ei välttämättä koeta tehokkaaksi ja XP:n käyttöönotto nähdään vaikeaksi organisaatiorakenteeltaan monimutkaisissa organisaatioissa (Dybå ja Dingsøyr, 2008). Menetelmän laaja suosio itsessään tukee XP:n myönteisiä kokemuksia käytännön ohjelmistokehityksessä. On kuitenkin selvää, että XP ei sovi jokaiseen tilanteeseen ja menetelmän käytännön rajoitteet eivät ole yksiselitteisiä (Beck, 1999a). XP tulisikin omaksua vähitellen, jolloin sen käyttöönottoon liittyviä riskejä voidaan pienentää (Beck, 1999a). XP:ssä esitetyt käytänteet voivat olla hyödyllisiä myös sellaisenaan: Paulkin (2001) mukaan ohjelmisto-

projektien tulisi omaksua XP:n arvoja riippumatta siitä, kuinka paljon ne eroavat organisaation olemassa olevista käytännöistä.

3.3 Scrum

Scrum (Schwaber & Beedle, 2002 ; Schwaber & Sutherland, 2013) on suosittu ketterän ohjelmistokehityksen viitekehys, joka on saanut vaikutteita teollisuuden prosessien hallinnasta. Scrum esittää joustavuuden, mukautuvuuden sekä tuottavuuden ajatusmallit ohjelmistokehityksen yhteydessä (Schwaber & Beedle, 2002). Alun perin Scrum esiteltiin menetelmänä (Schwaber, 1995), joka sisältää prosessin, käytänteet sekä joukon sääntöjä. Myöhemmin (Schwaber & Sutherland, 2013) Scrumin on sanottu olevan viitekehys monimutkaisten tuotteiden kehittämiseen sekä ylläpitoon. Scrum ei tarjoa toteutusvaiheen tekniikoita, vaan se keskittyy tiimin toimintaan ja järjestelmän joustavaan kehittämiseen jatkuvasti muuttuvassa ympäristössä (Abrahamsson ym., 2002).

Scrumin prosessi on esitetty seuraavaksi Abrahamssonin ym. (2002) mukaan, perustuen Schwaberin (1995) sekä Schwaberin ja Beedlen (2002) julkaisuihin. Prosessissa on kolme vaihetta (kuvio 7), jotka ovat esipeli (engl. pre-game), kehitys (engl. development) ja loppupeli (engl. post-game).



KUVIO 7 Scrum prosessi (Abrahamsson ym., 2002, s.28)

Esipelissä on kaksi alivaihetta, suunnittelu (engl. planning) ja arkkitehtuuri (engl. architecture/high level design). *Suunnitteluvaiheessa* laaditaan *tuotteen kehitysjono* (engl. product backlog), joka on priorisoitu lista järjestelmän vaatimuksista. Esimerkkejä kehitysjonon kohteista ovat muun muassa kehitettävän järjestelmän eri toiminnot, havaitut viat sekä pyydetyt laajennukset järjestelmään. Tuotteen kehitysjono päivittyy jatkuvasti, ja uusien kohteiden lisäksi siitä voidaan poistaa vanhoja kohteita tai tarkentaa niitä. Suunnitteluvaihe sisältää tuotteen kehitysjonon lisäksi muita tehtäviä kuten tiimin kokoamisen, resurssien ja työkalujen määrittämisen sekä projektin riskien arvioinnin. (Abrahamsson ym., 2002.)

Arkkitehtuurivaiheessa laaditaan järjestelmän arkkitehtuuri ja korkean tason suunnitelma tuotteen kehitysjonon kohteista. Ehdotukset toteutukseen valittavista kohteista käydään läpi suunnitelmien *katselmoinnissa* (engl. design review meeting), ja arkkitehtuurivaiheessa valmistellaan myös esisuunnitelmat julkaisujen sisällöistä. (Abrahamsson ym., 2002.)

Esipeliä seuraavassa *toteutusvaiheessa* järjestelmän kehitys tapahtuu ns. *sprinteissä* (engl. sprints). Sprinteillä tarkoitetaan järjestelmän toiminnallisuuden kehittämistä inkrementaalisesti siten, että jokainen iteraatio sisältää perinteiset ohjelmistokehityksen vaiheet eli vaatimusmäärittelyn, analyysin, suunnittelun, kehityksen ja toimittamisen. Yksi sprintti kestää viikosta neljään viikkoon, ja siinä voi olla mukana myös useampi tiimi. Jokaisen sprintin lähtökohtana on *sprintin kehitysjono* (engl. Sprint Backlog), joka sisältää tuotteen kehitysjonosta valitut, sprintin aikana toteutettavat kohteet. Tuotteen kehitysjonosta poiketen sprintin kehitysjono säilyy muuttumattomana sprintin ajan. (Abrahamsson ym., 2002.)

Viimeisen vaiheen *loppupelissä* vaatimukset on täytetty ja julkaisuun ei voida lisätä uusia kohteita. Järjestelmälle tehdään integrointia, järjestelmätestausta sekä dokumentointia, ja se on valmis julkaistavaksi. (Abrahamsson ym., 2002.)

Kuten XP:ssä, myös Scrumissa on määritelty joukko rooleja ja niihin kuuluvia vastuita seuraavasti (Schwaber & Sutherland, 2013):

- *Kehitystiimi* (engl. development team) on itseohjautuva, tasavertainen tiimi, joka vastaa tuotteen kehitysjonossa olevien kohteiden muuttamisesta tuoteversioiksi. Kehitysvastuu on koko kehitystiimillä yhdessä, vaikka tiimin jäsenten työt painottuisivatkin eri osa-alueisiin.
- *Tuoteomistaja* (engl. product owner) on vastuussa tuotteen kehitysjonon julkaisusta ja sen hallinnasta. Tuoteomistaja vastaa niin tuotteen kuin kehitystiimin työn arvon maksimoimisesta, ja kehitystiimin tulee työskennellä vain hänen asettamiensa vaatimusten mukaisesti.
- *Scrum-mestari* (engl. Scrum master) vastaa projektin etenemisestä Scrumin käytänteiden ja sääntöjen mukaisesti. Scrum-mestari huolehtii myös tiimin työskentelyn tuottavuudesta puuttumalla mahdollisiin kehitystyötä haittaaviin esteisiin.
- *Scrum-tiimi* (engl. Scrum-team) koostuu Scrum-mestarista, tuoteomistajasta ja kehitystiimistä. Tiimin mallissa on pyritty optimoimaan jousta-

vuus, luovuus ja tuottavuus. Tiimi tuottaa tuoteversioita inkrementaalisesti, jolloin palautetilaisuudet voidaan maksimoida ja tuotteesta on aina saatavilla toimiva versio.

Roolien lisäksi Scrumissa on esitetty joukko erilaisia käytänteitä. *Sprintin suunnittelussa* (engl. Sprint planning) luodaan sprintin aikana tehtävien töiden suunnitelmat. Suunnittelu tehdään sprintin suunnittelupalaverissa, jonka kesto rajataan enintään kahdeksaan tuntiin kuukauden mittaisella sprintillä. Scrummestari huolehtii siitä, että osallistujat ymmärtävät tapahtuman tarkoituksen, ja ohjaa Scrum-tiimiä suunnittelupalaverin aikarajassa pysymisessä. (Schwaber & Sutherland, 2013.)

Päiväpalaverissa (engl. Daily Scrum) luodaan suunnitelmat seuraavan vuorokauden ajaksi, ja tapahtuma on enintään 15 minuutin mittainen. Päiväpalaverissa käydään läpi edellisen tapaamisen jälkeen tehdyt työt, ja pohditaan mitä tehdään seuraavaksi. Kehitystiimille päiväpalaveri mahdollistaa työn edistymisen tarkastelun kohti sprintin tavoitetta. Tapaamisessa keskustellaan myös mahdollisesti esille tulleista esteistä tai ongelmista. (Schwaber & Sutherland, 2013.)

Sprintin katselmointi (engl. Sprint review) on sprintin päätteeksi pidettävä epävirallinen, maksimissaan neljä tuntia kestävä tilaisuus, johon osallistuvat Scrum-tiimi ja tuoteomistajan kutsumat sidosryhmien edustajat. Katselmointitilaisuuden osallistujat arvioivat kehitettyä tuoteversiota ja päättävät seuraavista toimista. Tilaisuudessa keskustellaan muun muassa toteutuksessa hyvin menneistä asioista ja esiin tulleista ongelmista sekä niiden ratkaisuista. Katselmoinnissa tuotetaan tarkastettu tuotteen kehitysjono ja seuraavan sprintin todennäköiset kehitysjonon kohteet. (Schwaber & Sutherland, 2013.)

Sprintin retrospektiivi (engl. Sprint retrospective) on enintään kolme tuntia kestävä tilaisuus, joka pidetään sprintin katselmoinnin jälkeen, mutta kuitenkin ennen seuraavan sprintin suunnittelupalaveria. Tilaisuuteen osallistuu Scrum-tiimi, ja sen tarkoitus on muun muassa tarkastella edellisen sprintin kokemuksia yhteistyöhön, prosessiin ja työkaluihin liittyen. Tilaisuuden tarkoituksena on tunnistaa tärkeimmät parannuskohteet tai hyvin sujuneet asiat ja luoda suunnitelma Scrum-tiimin työskentelyn parantamiseksi. (Schwaber & Sutherland, 2013.)

Scrumin käytöstä on olemassa jonkin verran aiempaa tutkimusaineistoa. Schwaberin (2004) mukaan Scrum toimii hyvin kiireellisissä, organisaatiolle kriittisissä projekteissa ja se parantaa tiimien suoriutumista. Sutherland, Viktorov, Blount ja Putnikov (2007) tukevat väitettä Scrumin kokemuksilla globaalisti hajautetuissa tiimeissä, jossa Java-projektin tuottavuus nousi Sutherlandin ym. (2007) mukaan korkeimmalle tasolle mitä on koskaan raportoitu. Myös Paasivaara, Durasiewicz ja Lassenius (2009) raportoivat Scrumin käytön hyödylliseksi hajautetussa kehityksessä. Kniberg (2007) puolestaan raportoi onnistuneista kokemuksista Scrum-käyttöönotossa, jossa niin johto, kehittäjät kuin testaajatkin olivat tyytyväisiä vaikeasta markkinatilanteesta ja henkilöstövähennyksistä huolimatta. Niin ikään Salon ja Abrahamssonin (2008) kyselytutkimus raportoi

positiivisia kokemuksia Scrumista: 77 % vastaajista koki Scrumin käytön positiivisena sulautettujen järjestelmien kehittämisessä. On kuitenkin huomioitava, että edellä mainitut tulokset ovat vain yksittäisiä esimerkkejä Scrumin käytännön kokemuksista ja kuten Dybå ja Dingsøyr (2008) mainitsevat, menetelmää on tutkittu varsin vähän sen suosioon verrattuna. On myös selvää, että Scrumin omaksuminen vaatii asianmukaisen koulutuksen, mikäli sitä ei ole organisaatiossa ennen käytetty (Paasivaara ym., 2009) ja Scrum voi muuttaa tiimin olemassa olevia työtehtäviä merkittävästi (Abrahamsson ym., 2002).

3.4 Yhteenveto

Tässä luvussa esiteltiin ketterän kehittämisen arvot ja periaatteet sekä kaksi yleisintä ketterää menetelmää, XP ja Scrum. Ketterän kehittämisen piirteiksi todettiin muun muassa inkrementaalinen ja iteratiivinen kehittäminen, yhteistyöhakuisuus sekä mukautuvuus muutoksiin.

XP:stä ja Scrumista esiteltiin prosessit, roolit sekä käytänteet. XP todettiin helposti omaksuttavaksi menetelmäksi ja sen käyttökokemukset todettiin yleisesti myönteisiksi. Organisaatiorakenteeltaan monimutkaisiin organisaatioihin menetelmän käyttöönotto todettiin kuitenkin haasteelliseksi.

Scrum todettiin joustavaksi, mukautuvaksi sekä tuottavaksi viitekehykseksi, joka sopii myös monimutkaisiin projekteihin. Sen käyttökokemukset käytännön kehitystyössä todettiin myönteisiksi, mutta viitekehyksen käyttöönoton todettiin vaativan asianmukaisen koulutuksen organisaatiossa.

4 TURVALLISUUDEN SUUNNITTELU OSANA KETTERÄÄ JÄRJESTELMÄKEHITYSTÄ

Tässä luvussa tarkastellaan turvallisuuden suunnittelun ja ketterän lähestymistavan yhdistämistä. Aluksi pohditaan ketterän kehittämisen ja turvallisuuden suunnittelun yhdistämisen haasteita ja toisaalta tukevia piirteitä. Seuraavaksi esitellään tutkimusten haussa ja valinnassa käytetty menetelmä ja prosessi. Tämän jälkeen käsitellään turvallisuuden suunnittelun menetelmiä ja käytänteitä ketterässä kehittämisessä, perustuen aiheesta julkaistuihin esityksiin ja XP:n sekä Scrumin räätälöinteihin. Lopuksi esitetään lyhyt yhteenveto luvun keskeisestä sisällöstä. Luvun tarkoituksena on luoda perusteet seuraavalle luvulle, jossa esitettyjä menetelmiä vertaillaan keskenään.

4.1 Integroinnissa yleisesti esiintyviä haasteita ja hyötyjä

Ketterän kehittämisen ja turvallisuuden suunnittelun integrointia käsitellään alan kirjallisuudessa monesta eri näkökulmasta. Yhteistä näkemyksille on kuitenkin se, että integroinnissa on tunnistettavissa useita haasteita ja luonteeltaan joustava ketterä kehittäminen on osittain ristiriidassa turvallisuuden suunnittelun vaatimuksien kanssa. Yksi yleisesti esitetty ristiriita voidaan kiteyttää turvallisuuden suunnittelun ja ketterän kehittämisen *prosessien* eroihin. Monet turvallisuuden suunnittelun menetelmät ja standardit perustuvat iteratiivisen ja joustavan lähestymistavan sijaan vaiheittaiseen kehitykseen, mikä vaatii vakaan toimintaympäristön, jossa tuotetaan kattavia suunnitelmia jo ennen järjestelmän toteuttamista (Boström ym., 2006; Kongsli, 2006; Singhal & Singhal, 2011). Jos tarkastellaan esimerkiksi luvussa 2 esitettyjen SSE-CMM (ISO/IEC 21827, 2008) -standardin tai CC:n (CC, 2012) käytänteitä järjestelmäkehityksen näkökulmasta, on haasteet kehitysprosessille helppo ymmärtää. Jo alemmille turvallisuuden arvioinnin ja kyvykkyyden tasoille vaaditaan paljon suunnitelmia ja kuvauksia, mikä on ristiriidassa ketterän kehittämisen periaatteiden kanssa muun muassa

aikaisen ja jatkuvan toimituksen, yksinkertaisuuden sekä muuttuviin vaatimuksiin reagoinnin suhteen.

Dokumentaatiolla on keskeinen merkitys turvallisuuden suunnittelussa, ja dokumentoinnin puutteet ketterässä kehityksessä mainitaan kirjallisuudessa yleisesti integroinnin haasteena (Chivers, Paige, & Ge, 2005; Beznosov & Kruchten, 2004; Goertzel, Winograd, McKinley, Oh, Colon, McGibbon & Vienneau, 2007). Dokumentaatio liittyy läheisesti *turvallisuuden varmistamiseen*, joka Farrohan ja Farrohan (2011) määritelmän mukaan tarkoittaa varmistumista siitä, että järjestelmässä ei ole tarkoituksella tai vahingossa suunniteltuja haavoittuvuuksia. On selvää, että ilman riittäviä perusteita (esim. dokumentteja) myös turvallisuuden varmistaminen jää puutteelliseksi. Muun muassa Beznosov ja Kruchten (2004), Peeters (2005), sekä Mougouei, Sani ja Almas (2013) pitävät turvallisuuden varmistamista ketterässä kehityksessä riittämättömänä.

Yhtenä haasteena integroinnille on myös *keskittyminen toiminnallisiin vaatimuksiin*. Muun muassa Caon ja Rameshin (2008) sekä Goertzelin ym. (2007) mukaan ketterässä kehityksessä laiminlyödään ei-toiminnallisia vaatimuksia, kuten turvallisuutta. Toiminnallisuuden arvostaminen näkyy myös Agile Allianssin (2001) ketterän kehittämisen arvoissa, eikä ei-toiminnallisia ominaisuuksia erityisesti huomioida missään yleisesti käytetyssä ketterässä menetelmässä.

Osittain ei-toiminnallisiin vaatimuksiin liittyvä haaste, *turvallisuusvaatimusten jäljittäminen,* on Peetersin (2005) mukaan suurin este ketterän lähestymistavan käytölle turvallisuuskriittisissä projekteissa. Jos väitettä tarkastellaan esimerkiksi luvussa 3 esiteltyjen XP:n ja Scrumin näkökulmasta, ei turvallisuusvaatimusten jäljittämiseen ole muiden turvallisuuteen liittyvien käytänteiden tavoin osoitettu erityistä huomiota. Goertzelin ym. (2007) mukaan turvallisuuden suunnittelu vaatii myös erityisosaamista, jota ei ole riittävästi kehitystiimissä. *Turvallisuusosaaminen* voi luonnollisesti vaihdella tiimin mukaan, mutta varsinaista roolia tai osaamisen kehittämistä turvallisuuden suunnitteluun ei ketterän kehittämisen menetelmissä ole.

Vaikka turvallisuutta ei ole erityisesti huomioitu missään ketterässä menetelmässä (Ge ym., 2007), löytyy ketterästä lähestymistavasta kuitenkin turvallisuutta tukevia piirteitä. Peetersin (2005) mukaan *kommunikointi* sidosryhmien kanssa palvelee turvallisuuden varmistamista paremmin kuin muodollisuus. Kommunikointi on kiistatta yksi ketterän lähestymistavan vahvuuksista määriteltäessä järjestelmän toiminnallisuutta, joten on perusteltua väittää sen tukevan myös turvallisuuden varmistamista ja suunnittelua. Ketterien menetelmien *reagointi ympäristön muutoksiin* voidaan nähdä toisaalta integroinnin haasteena, mutta kuten Azham, Ghani ja Ithnin (2011) toteavat, toimintaympäristö on tärkeä osa turvallisuutta, ja sen muutokset todennäköisesti altistavat järjestelmän uusille uhkille. Ketteryys sopii muutoksiin reagointiin hyvin, minkä toteaa myös Peeters (2005): järjestelmän pitää mukautua ympäristön muutoksiin pysyäkseen turvallisena.

Turvallisuuspuutteiden asteittainen löytäminen ja niiden vastatoimien vähittäinen kehittäminen on Singhalin ja Singhalin (2011) mukaan yksi ketterän lä-

hestymistavan hyödyistä turvallisuuden suunnittelussa. Väitettä tukevat myös Boström ym. (2006), joiden mukaan iteratiivinen kehittäminen mahdollistaa nopean palautteen ja lisää turvallisuusvaatimusten prosessin tehokkuutta. Heidän mukaansa ketterien menetelmien yksinkertaiset dokumentoinnin tekniikat, kuten indeksikortit ja valkotaulupiirustukset, vähentävät turvallisuuden suunnittelussa tarvittavaa työmäärää.

Ketterä lähestymistapa täyttää hyvin sidosryhmien tarpeet asteittaisella kehityksellä ja on yleisesti hyväksytty, mutta turvallisuuden integrointiin ei ole tehty vastaavaa työtä (Chivers ym., 2005). Edellä käsiteltyjen yleisimpien haasteiden ja hyötyjen pohjalta on kuitenkin esitetty useita käytänteitä ja menetelmiä, joilla turvallisuus pyritään huomioimaan ketterien periaatteiden mukaisesti. Näitä esityksiä käsitellään tarkemmin alaluvuissa 4.3 ja 4.4, ensin yleisiin esityksiin perustuen ja lopuksi XP:n ja Scrumin räätälöinneistä.

4.2 Tutkimusmenetelmä ja –prosessi

Tässä tutkielmassa käytetään relevantin kirjallisuuden haussa ja valinnassa tutkimusmenetelmänä integroivaa kirjallisuuskatsausta (Torraco, 2005). Tutkimusmenetelmä mahdollistaa aihealueen tarkastelun monipuolisesti, ja sen avulla voidaan tuottaa hyvin uutta tietoa aikaisemmin tutkitusta aiheesta. Tutkimusprosessissa on käytetty Cooperin (1998) mukaisia tutkimusvaiheita. Mallia on yleisesti hyödynnetty erityisesti sosiaalitieteissä, mutta se on tieteenalasta riippumaton ja soveltuu hyvin myös tämän tutkielman tarkoituksiin.

Ensimmäisessä vaiheessa asetettiin tutkimusongelma, jonka tarkoituksena on Cooperin (1998) mukaan rakentaa määrittelyt, jotka erottavat olennaiset teokset epäolennaisista. Tutkimusongelma *"Miten turvallisuutta voidaan suunnitella osana ketterää järjestelmäkehitystä?"* pitää sisällään kaksi tärkeää osa-aluetta, turvallisuuden suunnittelun ja ketterän järjestelmäkehityksen. Suuri osa kirjallisuuskatsauksen julkaisuista käsittelee sekä turvallisuuden suunnittelua että ohjelmisto- tai järjestelmäkehitystä. Toisaalta tarkentavilla tutkimuskysymyksillä voitiin rajata riittävästi tutkimusongelmaa, jolloin tarkastelun kohteena olivat vain relevantit julkaisut.

Toisessa vaiheessa toteutettiin aineiston haku, joka aloitettiin Cooperin (1998) menetelmän mukaisesti määrittelemällä relevantit tietolähteet ja tiedonhankinnan menetelmät. Tutkielman ensisijaiseksi tiedonlähteeksi valittiin sähköisessä muodossa oleva aineisto, ja tiedonhaussa käytettiin muun muassa seuraavia tietokantoja:

- Springer
- IEEE Xplore Digital Library
- ACM Digital Library
- CiteSeerX
- ScienceDirect

Lisäksi tietoa haettiin Internet-lähteistä ja alan kirjallisuudesta. Sähköisten tietokantojen hakusanoina käytettiin lukuisia ketterän kehittämisen ja turvallisuuden suunnittelun asiasanoja sekä näiden yhdistelmiä, kuten *"agile development"*, *"security engineering"*, *"security in agile development"*, *"agile security"* tai *"secure agile development"*. Cooperin (1998) menetelmässä aineiston arviointi on erillinen vaihe, ja sen tarkoituksena on muodostaa kriteerit, jolla relevantti aineisto erotetaan muusta aineistosta. Tämä vaihe yhdistettiin aineiston hakuun, sillä osa aineistosta karsittiin jo hakuvaiheessa otsikon ja tiivistelmän perusteella. Kriteerinä käytettiin tutkimuksen viitekehystä (alaluku 5.1), jonka mukaan julkaisussa tuli olla huomioituna ketterän kehittämisen ja turvallisuuden suunnittelun integroinnin prosessi tai jokin sen osa-alue, roolit ja tarvittavat käytänteet sekä tuotokset. Jos julkaisu käsitteli esimerkiksi ohjelmistokehityksen ja turvallisuuden integrointia yleisesti, mutta ei ketterää kehittämistä, rajattiin se tarkastelun ulkopuolelle. Myös ketterää kehittämistä ja turvallisuuden suunnittelua käsittelevät julkaisut, joissa integrointia käsiteltiin liian yleisellä tasolla, rajattiin tarkastelun ulkopuolelle. Lisäksi seuraavien ehtojen tuli täyttyä:

- Julkaisu on julkaistu 2000-luvulla.
- Julkaisun kieli on englanti tai suomi.
- Julkaisu on väitöskirja, lisensiaattityö tai muu julkaisu, jolle on tehty vertaisarviointi.

Seuraavassa vaiheessa kerättyä aineistoa tarkasteltiin sisällöllisesti ja valittiin tutkielmassa käsiteltävät julkaisut. Cooperin (1998) mukaan sisällöllinen analyysi ja esitysten valinta ovat kaksi erillistä vaihetta, mutta tässä tutkielmassa vaiheet yhdistettiin. Tutkielmassa käsiteltävistä julkaisuista tarkempaan kuvaukseen valittiin lopulta yhdeksän menetelmää, jotka on esitetty taulukossa 3.

TAULUKKO 3 Yksityiskohtaiseen tarkasteluun valitut julkaisut

Tekijät	**Julkaisun nimi**
Azham, Ghani & Ithin, 2011	Security Backlog in Scrum Security Practices
Baca & Carlsson, 2011	Agile Development with Security Engineering Activities
Beznosov, 2003	Extreme Security Engineering: On Employing XP Practices to Achieve "Good Enough Security" without Defining It
Boström, Wäyrynen & Bodén, 2006	Extending XP Practices to Support Security Requirements Engineering
Kongsli, 2006	Towards Agile Security in Web Applications
Mougouei, Sani & Almasi, 2013	S-Scrum: a Secure Methodology for Agile Development of Web Services
Peeters, 2005	Agile Security Requirements Engineering
Singhal & Singhal, 2011	Development of Agile Security Framework Using a Hybrid Technique for Requirements Elicitation
Siponen, Baskerville & Kuivalainen, 2005	Integrating Security into Agile Development Methods

Tutkielmassa käsiteltävät julkaisut valittiin tutkimuksen viitekehykseen (alaluku 5.1) ja laadulliseen analyysiin perustuen. Niistä pyrittiin löytämään yhteisiä piirteitä, ja ristiriitaisuuksille etsittiin täydentävää aineistoa sähköisten tietokantojen täsmennetyillä hauilla.

4.3 Esitettyjä menetelmiä ja käytänteitä

Seuraavaksi käsitellään turvallisuuden suunnittelun ja ketterän kehittämisen yhdistäviä käytänteitä ja menetelmiä, jotka eivät perustu mihinkään aiempaan ketterään menetelmään tai viitekehykseen. Ensiksi esitellään väärinkäytön kertomuksia, jonka jälkeen käsitellään ketterän turvallisuuden viitekehyksen yleispiirteet. Tämän jälkeen käsitellään kahta menetelmää, joissa ketterään kehittämiseen yhteensopivat turvallisuuden suunnittelun aktiviteetit on sisällytetty osaksi kehittämisen osa-alueita. Lopuksi esitellään turvallisen ja ketterän kehityksen Web-prosessi.

4.3.1 Väärinkäytön kertomukset

Ketterässä kehittämisessä käytetään yleisesti käyttäjäkertomuksia järjestelmän toiminnallisuuksien kuvaamiseen. Peeters (2005) on kehittänyt käyttäjäkertomuksista turvallisuuden suunnitteluun räätälöidyn version, *väärinkäytön kertomukset* (engl. abuser stories). Väärinkäytön kertomusten periaate on vastaava kuin alaluvussa 2.4 esitellyissä väärinkäyttötapauksissa; lähtökohdaksi otetaan ohjelmistokehityksen yleinen käytänne, jota sovelletaan uudella tavalla turvallisuuden suunnitteluun.

Turvallisuusvaatimuksia voidaan Peetersin (2005) mukaan mallintaa tiettyyn tasoon saakka tavallisilla käyttäjäkertomuksilla. Esimerkkinä hän mainitsee Web-vedonlyönnin sovelluksen, jossa tavallisen käyttäjäkertomuksen lause näyttäisi tältä (Peeters, 2005, s.2):

> ...Käyttäjä täyttää vedon summan ja pelaa…

Jotta käyttäjäkertomuksen toiminnallisuus tuottaisi riittävästi arvoa liiketoiminnalle, tulee sen sisältää myös käyttäjän todentaminen:

> …Käyttäjä todentaa itsensä salasanalla. Hän täyttää vedon summan ja pelaa…

Jälkimmäisessä kertomuksessa on huomioitu vastatoimi sille, että joku esiintyy sovelluksessa vieraana käyttäjänä. Kaikkia uhkia, kuten esimerkin pelissä käytettävän satunnaisgeneraattorin manipulointia, ei kuitenkaan voida tällä tavoin ilmaista, jolloin käyttäjäkertomuksia on laajennettava väärinkäytön kertomuksilla kuvaamaan paremmin mahdollista uhkaa. (Peeters, 2005, s.2.)

Tavalliset käyttäjäkertomukset pisteytetään niiden liiketoiminnallisen arvon perusteella. Väärinkäytön kertomuksissa periaate on sama, mutta arvo on

käänteinen; järjestelmän väärinkäytön tai hyökkäyksen arvo on haitallinen liiketoiminnalle. Pisteytyksessä on tärkeää huomioida väärinkäytön kertomuksen vahingon laajuus ja onnistuneen hyökkäyksen todennäköisyys. Väärinkäytön kertomusten pisteet tulisi olla myös samalla tasolla tavallisten käyttäjäkertomusten pisteisiin. Kuten tavallisissa käyttäjäkertomuksissa, myös väärinkäytön kertomuksissa pisteytykset voivat muuttua ympäristön muutosten, kuten teknologisen kehityksen seurauksena. (Peeters, 2005.)

Työmäärän arviointi on tärkeä osa järjestelmän suunnittelua, ja arviointi on toteutettu eri tavoin käytettävästä menetelmästä riippuen. Esimerkiksi XP:ssä käyttäjäkertomusten vaatima työmäärä arvioidaan suunnittelupelissä (Beck, 1999a) ja sama käytänne sopii vastaavasti väärinkäytön kertomuksiin. Näkökulma on kuitenkin jälleen mahdollisessa uhkassa: työmääräarviot sisältävät väärinkäytön kertomuksessa kuvatun uhkan torjumiseen tarvittavan vastatoimen työmäärän (Peeters, 2005). Tällaisella vastatoimella voidaan tarkoittaa esimerkiksi käyttäjän todennusta, salausalgoritmin kehittämistä tai muuta järjestelmän toiminnallisuutta, jolla mahdolliset uhkat voidaan torjua. Siinä missä hyväksyntätestaus osoittaa onnistuneen toteutuksen käyttäjäkertomuksissa, on uhkien kumoaminen keskeistä väärinkäytön kertomuksissa (Peeters, 2005). Järjestelmää ei voida kehittää täysin turvalliseksi, mutta uhkien kumoamisella voidaan todistaa, että kuvatut uhkat eivät toteudu tai niiden toteutuminen on epätodennäköistä (Peeters, 2005). Peeters (2005) ei mainitse käytännön esimerkkiä siitä, kuinka uhkien kumoaminen osoitetaan. Esimerkiksi käyttäjän todentaminen voidaan esittää normaalilla hyväksyntätestauksella ja lyhyellä kuvauksella toteutetusta todennusmenetelmästä. Sen sijaan ohjelmiston sisäistä turvallisuutta lisäävää mekanismia voi olla mahdotonta todentaa, koska torjuttava uhka ei ole välttämättä ennalta tiedossa. Keskeistä Peetersin (2005) mainitsemassa uhkien kumoamisessa lienee joka tapauksessa se, että mahdollinen turvallisuusuhka on huomioitu järjestelmän toiminnallisuudessa ja se voidaan tavalla tai toisella osoittaa.

Tavallisissa käyttäjäkertomuksissa asiakkaat laativat kertomukset (Abrahamsson ym., 2002; Peeters, 2005). Asiakkaan tulee olla Peetersin (2005) mukaan laatimassa myös väärinkäytön kertomuksia, sillä asiakas tuntee parhaiten suojattavat kohteet. Peeters (2005) esittää kuitenkin myös kehittäjille roolia väärinkäytön kertomusten laadinnassa, sillä uhka voi olla luonteeltaan tekninen. Esimerkkinä hän käyttää jälleen Web-uhkapeliä, jossa asiakas painottaa saavutettujen voittojen suojaamisen tärkeyttä ja kehittäjä puolestaan näkee uhkana peliprosessin manipuloinnin teknisesti. Näin molempien asiantuntemusta voidaan hyödyntää kertomusten laadinnassa. Toisaalta sekä asiakkaan että kehittäjien osallistuminen väärinkäytön kertomusten laadintaan vahvistaa ketterän kehittämisen periaatteita tiimin jäsenten yhteistyöstä (Peeters, 2005).

Suojattavat pääomat ovat hyvä lähtökohta väärinkäytön kertomusten kirjoittamiselle. Kaikkea pääomaa, jolla on asiakkaalle arvoa ja jota voidaan mahdollisesti käyttää järjestelmän kautta, tulisi käsitellä mahdollisena kohteena. (Peeters, 2005.) Tämä on loogista, sillä pohjimmiltaan turvallisuuden suunnittelulla pyritään järjestelmään liittyvien pääomien, kuten luottamuksellisen tiedon

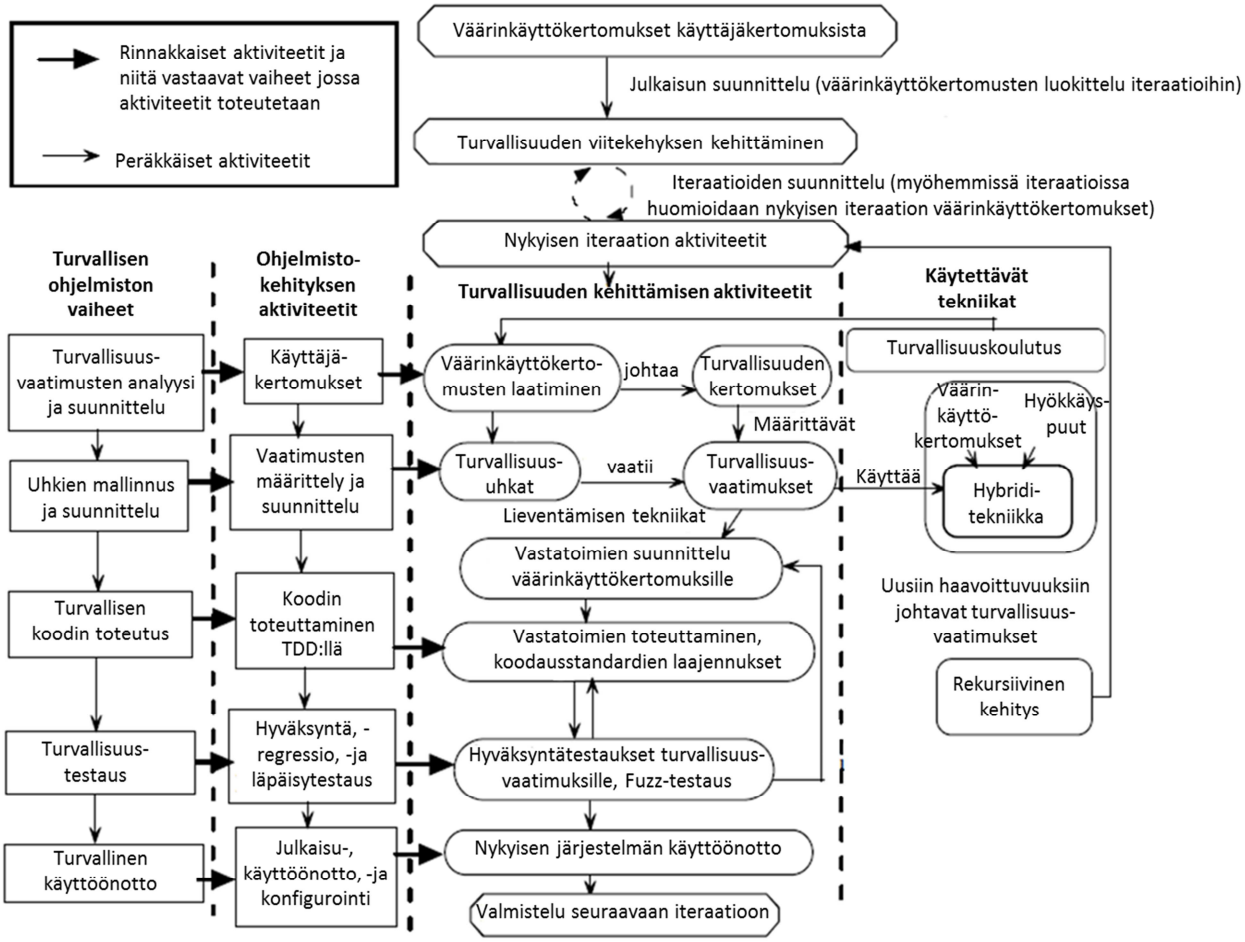
tai omaisuuden suojaamisen. Kertomuksia laadittaessa on toisaalta tärkeää tarkastella myös mahdollisten järjestelmän väärinkäyttäjien piirteitä, sillä usein ne heijastuvat järjestelmää kohtaavaan uhkaan (Peeters, 2005). Keskeisiä tekijöitä tässä ovat muun muassa resurssit, taidot ja motivaatiot, jotka mainitaan tärkeiksi ominaisuuksiksi myös McDermotin ja Foxin (1999) väärinkäyttötapausten mallintamisessa.

4.3.2 Agile Security Framework (ASF)

Singhal ja Singhal (2011) esittävät turvallisuuden suunnitteluun koko järjestelmän kehitysprosessin kattavaa ketterän turvallisuuden viitekehystä (engl. Agile Security Framework, ASF), jossa eri menetelmien piirteitä on yhdistetty samaan viitekehykseen. Singhalin ja Singhalin (2011) viitekehyksessä esitetään käytettäväksi turvallisuuden suunnitteluun väärinkäytön kertomusten ja hyökkäyspuiden (Schneier, 1999) (ks. alaluku 2.5) yhdistelmää, jota kutsutaan *hybriditekniikaksi* (engl. hybrid technique). Ketterän kehittämisen periaatteet on puolestaan huomioitu siten, että turvallisuuden suunnittelun vaatimat muutokset kehitysprosessissa eivät vaikuta ketterän lähestymistavan piirteisiin, kuten iteratiiviseen ja inkrementaaliseen kehittämiseen tai joustavuuteen muutoksissa.

ASF koostuu järjestelmän kehityksen eri vaiheista sekä rinnakkain suoritettavista aktiviteeteista (ks. kuvio 8). Vaiheet on nimetty *turvallisen ohjelmiston vaiheiksi* (engl. secure software phases) ja aktiviteetit *ohjelmistokehityksen aktiviteeteiksi* (engl. software development activities) sekä *turvallisuuden kehittämisen aktiviteeteiksi* (engl. security development activities). Lisäksi viitekehys sisältää aktiviteeteissa hyödynnettävät tekniikat, jotka ovat hybriditekniikka, turvallisuuskoulutus ja rekursiivinen kehittäminen. (Singhal & Singhal, 2011.) Vaiheiden kanssa rinnakkain suoritettavat aktiviteetit yhdistävät turvallisuuden suunnittelun ja ohjelmistokehityksen samankaltaiset käytänteet toisiinsa. Esimerkiksi käyttäjäkertomukset ja väärinkäytön kertomukset laaditaan samassa vaiheessa siten, että väärinkäytön kertomukset perustuvat käyttäjäkertomuksiin. Tämä tukee myös Peetersin (2005) näkemystä väärinkäytön kertomuksista, sillä hänen mukaansa käyttäjäkertomusten ja väärinkäytön kertomusten tulisi olla arvoltaan yhteneväiset. Myös muut aktiviteetit, kuten vaatimusmäärittely ja testaus, suoritetaan samanaikaisesti ohjelmistokehityksen aktiviteettien ja turvallisuuden kehittämisen aktiviteettien osalta. Turvallisuus on integroitava osaksi kehitystyötä alusta alkaen (Farroha & Farroha, 2011; Singhal & Singhal, 2011), mikä on otettu viitekehyksessä hyvin huomioon.

Ensimmäinen viitekehyksen vaihe sisältää *turvallisuusvaatimusten analyysin ja suunnittelun* (engl. security requirements analysis and planning), ja sen tarkoituksena on määrittää kriittiset kohteet sekä muodostaa väärinkäytön kertomukset. Kriittisten kohteiden määrittämisessä kehittäjät analysoivat käyttäjäkertomuksia, tunnistavat järjestelmän suojattavat kohteet sekä määrittävät turvallisuustavoitteet ja mahdolliset rajoitteet suhteessa käyttäjäkertomuksiin. Väärinkäytön kertomukset muodostetaan kuvaamalla järjestelmän sellainen toiminnallisuus, jonka ei haluta toteutuvan. Kertomuksissa käytetään pisteytet



KUVIO 8 ASF-viitekehyksen rakenne (Singhal & Singhal, 2011, s.5)

tyjä kortteja, ja niiden tekemiseen osallistuvat kehitettävän järjestelmän sidosryhmät, kehittäjät ja turvallisuuden suunnittelun asiantuntijat. (Singhal & Singhal, 2011.) Väärinkäytön kertomukset muodostetaan siis vastaavasti kuin Peetersin (2005) esityksessä asiakkaan ja kehittäjien yhteistyönä, mutta Singhal ja Singhal (2011) käyttävät myös turvallisuuden asiantuntijoita kertomusten laatimiseen. Lisäksi he painottavat turvallisuuskoulutuksen merkitystä väärinkäytön kertomusten laadinnassa yhdenmukaisesti Gen ym. (2007) esityksen kanssa.

Ensimmäisessä iteraatiossa luodaan turvallisuuden viitekehys, jossa väärinkäytön kertomukset luokitellaan niiden vakavuuteen, vaikutusalueeseen, riskitekijöihin sekä käyttäjätarpeisiin perustuen. Luokittelun avulla kehittäjät ymmärtävät paremmin, mikä kertomus toteutetaan missäkin iteraatiossa. Näin esimerkiksi uhkien mallintaminen voidaan tehdä vain nykyisen iteraation kertomuksista. Ensimmäisessä iteraatiossa toteutetaan myös julkaisun suunnittelu, jonka periaate on sama kuin Leffingwellin, Jemilon, Zamoran, O´Neillin ja Yakuman (2014) SAFE-viitekehyksessä. Julkaisun suunnittelussa määritellään koko projektin toiminnallisuus sisältäen kaikki turvallisuuden vaatimukset. Iteraatio on muita pidempi, sillä turvallisuustoimintojen suunnittelu ja luokittelu vie aikaa. Vaihtoehtoisesti toinen iteraatio voidaan keskittää täysin turvallisuu-

delle, jolloin iteraatioista voidaan tehdä samanpituisia. (Singhal & Singhal, 2011.)

Toinen vaihe koostuu *uhkien mallintamisesta ja suunnittelusta* (engl. threat modelling and designing). Vaihe on Singhalin ja Singhalin (2011) mukaan erittäin tärkeä, ja sen aktiviteetit on siksi tehtävä huolellisesti. Vaiheeseen kuuluu yhtenä osana riskien arviointi ja priorisointi. Tarkastelun perustana ovat väärinkäytön kertomukset, ja riskit priorisoidaan niiden järjestelmävaikutusten perusteella. Suurimman riskin muodostavat kertomukset toteutetaan ensin. Turvallisuusvaatimusten esillesaanti kuuluu myös yhtenä kokonaisuutena uhkien mallintamisen ja suunnittelun vaiheeseen. Tässä hyödynnetään väärinkäytön kertomusten ja hyökkäyspuiden yhdistelmää, hybriditekniikkaa, jolla molempien tekniikoiden parhaat puolet voidaan yhdistää. Singhalin ja Singhalin (2011) mukaan väärinkäytön kertomuksilla ei välttämättä voida tunnistaa mahdollisen hyökkäyksen toteutustapaa, toisin sanoen järjestelmän sisäänpääsyn ja poistumisen pistettä, johon hyökkäyspuut sopivat paremmin. Toisaalta hyökkäyspuut eivät määrittele uhkien välttämisen strategiaa tai uhkan hetkellä vallitsevaa tilannetta, joka voidaan kuvata väärinkäytön kertomuksilla. Turvallisuusvaatimusten esillesaannin lisäksi uhkien mallintamisen ja suunnittelun vaiheeseen kuuluu turvallisuusvaatimusten suunnittelu. Tällä tarkoitetaan kaikkien tunnistettujen turvallisuusvaatimusten mallintamista. (Singhal & Singhal, 2011.) Ketterän lähestymistavan yhtenä periaatteena on Agile Allianssin (2001) mukaan yksinkertaisuus, joka on toisaalta ristiriidassa kaikkien turvallisuusvaatimusten mallintamiseen. Singhal ja Singhal (2011) kuitenkin painottavat, että mallintaminen tulee tehdä mahdollisimman yksinkertaisesti. He käyttävät esimerkkinä käyttötapausta muistuttavaa kaaviota, jossa mallinnetaan eri symbolein uhkan esivaatimukset, suojattavat kohteet, uhkat sekä tekniikat, joilla uhkat voidaan torjua. Mallintaminen antaa siis käytännössä perusteet tarvittaville vastatoimille.

ASF:n kolmas vaihe koostuu *turvallisen koodin toteutuksesta* (engl. secure code implementation). Tässä vaiheessa turvallisuusvaatimukset ja vastatoimet toteutetaan kuten muussa kehitystyössä. Toteutus on testauslähtöistä (engl. test driven development, TDD), joten yksikkötestit liitetään osaksi sprintin tai iteraation turvallisuuskertomuksia. Vaiheen lopputuloksena saadaan lopullinen toteutus järjestelmästä. (Singhal & Singhal, 2011.)

Neljännessä vaiheessa tehdään *turvallisuustestaus* (engl. security testing). Vaikka testaus on yksi ASF:n vaiheista, Singhal ja Singhal (2011) korostavat, että testausta tulee suorittaa yksikkötestien ja hyväksyntätestien osalta jatkuvasti. Turvallisuuteen liittyvät yksikkötestit luodaan ennen toteutusta, kuten tavallisissa yksikkötesteissä. Myös hyväksyntätestit luodaan vastaavasti kuin tavallisissa käyttäjäkertomuksissa, mutta ne perustuvat väärinkäytön kertomuksiin. Varsinaisessa turvallisuustestauksen vaiheessa tehdään niin sanottuja Fuzz-testejä, joissa järjestelmään syötetään viallista ja odottamatonta dataa järjestelmän vakauden testaamiseksi. Myös läpäisytestaus (engl. penetration testing) on yksi testausvaiheen osa-alueista, ja sillä pyritään löytämään haavoittuvuuksia erityisesti monimutkaisemmista sovelluksista. (Singhal & Singhal, 2011.)

Viimeinen ASF:n vaihe on *turvallinen käyttöönotto* (engl. Secure Deployment), jossa järjestelmä on valmis julkaisuun. Käyttöönotossa tarkkaillaan mahdollisia haavoittuvuuksia, ja se sisältää myös seuraavan julkaisun valmisteluun liittyvät toimenpiteet. (Singhal & Singhal, 2011.)

4.3.3 Turvallisuuden aktiviteeteilla laajennettu ketterä kehitys

Baca ja Carlsson (2011) analysoivat Common Criterian (CC, 2012), Microsoft SDL:n (Howard & Lipner, 2009) ja Cigatel Touchpoints:n (McGraw, 2006) turvallisuuden suunnittelun aktiviteetteja ja esittävät niitä soveltuvin osin laajennettavaksi ketterän kehittämisen käytänteisiin. Baca ja Carlsson (2011) käyttävät tausta-aineistona mainittujen menetelmien lisäksi suuren tietoliikennealan yrityksen kehittäjien haastatteluita. Esitettyjen käytänteiden tavoitteena on tunnistaa ketterään kehitykseen helposti integroitavat turvallisuuden suunnittelun käytänteet ja toisaalta sellaiset osa-alueet, joita ei ole mahdollista toteuttaa osana ketterää kehitystä (Baca & Carlsson, 2011). Seuraavaksi käsitellään Bacan ja Carlssonin (2011) esityksestä ketterän kehittämisen kanssa yhteensopivat käytänteet. Baca ja Carslsson (2011) esittelevät aktiviteetteihin liittyen myös tutkimuksen kohteena olevan yrityksen kehittämisprosessin. Prosessin tarkastelu jätetään kuitenkin tämän tutkielman ulkopuolelle, sillä esitetyt turvallisuuden suunnittelun käytänteet käsitellään suurelta osin prosessista erillään, ja toisaalta käytänteitä voidaan soveltaa ketterässä kehityksessä käytetystä prosessista riippumatta. Käytänteitä käsitellään kuitenkin Bacan ja Carlssonin (2011) yleiskäyttöisen prosessin kannalta, joka sisältää kehittämisen vaiheet vaatimusmäärittelystä testaukseen.

Taulukossa 4 esitetään käytänteet ja niiden liittyminen kehitysprosessin vaiheisiin sekä rooleihin. Vastuut jakautuvat järjestelmän vaatimusmäärittelyssä tuoteomistajalle ja suunnittelussa, toteutuksessa ja julkaisussa kehitystiimille. Testauksesta vastaa viimeisimmän version testaustiimi. Baca ja Carlsson (2011) huomauttavat, että kaikki turvallisuuden suunnittelun aktiviteetit eivät ole mukana, mutta valituilla aktiviteeteilla voidaan luoda ketterä ja turvallinen prosessi, joka toteuttaa perustana olevan kolmen turvallisuuden suunnittelun menetelmän parhaat puolet. Bacan ja Carlssonin (2011) mukaan järjestelmän vaatimusmäärittelyssä on kaksi tuoteomistajan vastuulla olevaa, ketterään kehitykseen yhteensopivaa turvallisuuden suunnittelun aktiviteettia, jotka ovat *turvallisuusvaatimukset* (engl. security requirements) ja *roolimatriisi* (engl. role matrix). Ensin mainittu perustuu McGrawn (2006) esitykseen, ja sillä tarkoitetaan vaatimusmäärittelyä, joka kattaa järjestelmän toiminnallisen turvallisuuden lisäksi esille nousevat uudet ominaisuudet, jotka voidaan tunnistaa parhaiten väärinkäyttötapauksilla ja hyökkäyskaavioilla. Roolimatriisi on puolestaan johdettu Microsoft SDL:stä (Howard & Lipner, 2009), ja menetelmällä voidaan tunnistaa kaikki mahdolliset käyttäjäroolit ja niiden pääsyoikeus järjestelmään. Keskeistä on tuottaa täsmällisiä turvallisuusvaatimuksia, jotka auttavat kehittäjiä ja testaajia ymmärtämään vaadittuja turvallisuustavoitteita. Roolimatriisien avulla

käyttäjäkertomukset voidaan määritellä turvallisuuden kannalta tarkemmin, mikä auttaa järjestelmän turvallisessa suunnittelussa. (Baca & Carlsson, 2011.)

TAULUKKO 4 Turvallisen ja ketterän kehittämisen aktiviteetit (Baca & Carlsson, 2011, s.8)

Tuoteomistaja	
Vaatimusmäärittely	
Turvallisuusvaatimukset	Roolimatriisi

Kehitystiimi	
Suunnittelu	Toteutus
Vastatoimien kaaviot	Staattisen koodin analyysit
Oletusten dokumentaatio	Koodaussäännöt
Väärinkäyttötapaukset	
Vaatimusten tarkastus	
Julkaisu	
Repositorin parantaminen (retrospektiivi tapaaminen)	

Testaustiimi
Testaus
Dynaamiset analyysit

Kehitystiimi toteuttaa järjestelmän suunnitteluun liittyvät turvallisuuden suunnittelun aktiviteetit, joita ovat *vastatoimien kaaviot* (engl. countermeasure graphs), *oletusten dokumentaatio* (engl. assumption documentation), *väärinkäyttötapaukset* (engl. abuse cases) sekä *vaatimusten tarkastus* (engl. requirements inspection) (Baca & Carlsson, 2011). Vastatoimien kaavioilla tarkoitetaan Bacan ja Petersenin (2010) esitykseen perustuvaa riskianalyysin menetelmää, joka keskittyy turvallisuustoimintojen tunnistamiseen sekä niiden priorisointiin. Oletusten dokumentaatio sisältää suunnittelijoiden, arkkitehtien ja analysoijien tunnistamat mahdolliset hyökkäykset heidän tekemiensä oletusten perusteella. Aktiviteetti perustuu McGrawn (2006) esitykseen. Väärinkäyttötapauksien McDermotin ja Foxin (1999) mukainen esitys on kuvattu luvussa 2.4, mutta Baca ja Carlsson (2011) viittaavat kuitenkin myös väärinkäyttötapauksissa Mcrawn (2006) esitykseen, jonka mukaan ne kuvaavat järjestelmän käyttäytymistä hyökkäyksen alaisuudessa. Aktiviteetin perusperiaate on kuitenkin sama valitusta väärinkäyttötapausten menetelmästä riippumatta. Vaatimusten tarkastuksella tarkoitetaan kaikkien tuotettujen artefaktien katselmointia, ja aktiviteetti perustuu Common Criteriaan (CC, 2012). Siinä tuotetaan tarkastusraportti (engl. validation report), jossa katselmoidaan tiimin työskentelyn laatua ja arvioidaan turvallisuusvaatimusten prosessia (Baca & Carlsson, 2011).

Kehitystiimin turvallisuuden aktiviteetteja ovat myös järjestelmän toteutuksen *staattisen koodin analyysit* (engl. static code analyses) ja *koodaussäännöt* (engl. coding rules). Ensin mainitulla tarkoitetaan koodin tarkastelua ja kooditasolla virheiden tunnistamista esimerkiksi soveltuvilla työkaluilla. Koodaus-

säännöt auttavat turvattomien toimintojen tunnistamisessa ja niiden vaihtoehtoisissa toiminnoissa. Staattisen koodin analyysi perustuu McGrawn (2006) vastaavaan aktiviteettiin ja koodaussäännöt puolestaan Microsoft SDL:ään (Howard & Lipner, 2009).

Baca ja Carlsson (2011) nimeävät julkaisuvaiheeseen yhden turvallisuuden suunnittelun aktiviteetin, *repositorin parantamisen* (engl. repository improvement). Aktiviteetti perustuu Common Criteriaan (CC, 2012), ja sen toteuttaa kehitystiimi. Repositorin parantamisessa esitellään järjestelmän uudet mallintamisen elementit (engl. model elements), joita käytetään mahdollisesti tulevissa sovelluksissa. Lisäksi repositorin olemassa olevia elementtejä voidaan muokata niiden laadun parantamiseksi. (Baca & Carlsson, 2011.)

Järjestelmän testauksen suorittaa testaustiimi, ja siinä suoritetaan yksi turvallisuuden suunnittelun aktiviteetti, *dynaamiset analyysit* (engl. dynamic analyses). Microsoft SDL:ään (Howard & Lipner, 2009) perustuvassa aktiviteetissa käytetään automaattisia testaustyökaluja ja suoritetaan testausten arviointia. Bacan ja Carlssonin (2011) mukaan monet turvallisuuden suunnittelun menetelmät keskittyvät manuaaliseen testaukseen ja ainoastaan dynaamiset analyysit arvostetaan korkealle, koska ne eivät vaadi manuaalista työtä.

Baca ja Carlsson (2011) ovat tehneet Common Criterian (CC, 2012), Microsoft SDL:n (Howard & Lipner, 2009) ja Cigatel Touchpoints:n (McGraw, 2006) valittuihin käytänteisiin joitain muutoksia, jotta ne tukevat paremmin ketterän kehittämisen prosessia. Huomionarvoista on väärinkäyttötapausten laatiminen kehittäjävetoisesti vasta suunnitteluvaiheessa. Myös repositorin parannus sopii Bacan ja Carlssonin (2011) mukaan paremmin myöhemmän vaiheen retrospektiiviin tapaamisiin.

4.3.4 Turvallisen ketterän kehityksen prosessi ja -elementit

Siponen, Baskerville ja Kuivalainen (2005) esittävät ketterään kehittämiseen sovellettavan turvallisuuden suunnittelun prosessin. Heidän mukaansa turvallisuus voidaan integroida osaksi ketterää kehitystä, mutta käytettävien tekniikoiden tulee täyttää tiettyjä vaatimuksia yhteensopivuuden varmistamiseksi. Seuraavaksi käsitellään nämä vaatimukset sekä Siposen ym. (2005) esittämä ketterään kehittämiseen soveltuva prosessi.

Turvallisuuden suunnittelun menetelmille ketterässä kehittämisessä tunnistetaan neljä vaatimusta, jotka on lueteltu alla (Siponen ym., 2005, s.1):

1. Turvallisuuden suunnittelun menetelmän mukautuvuus ketterän kehittämisen menetelmiin
2. Menetelmien täytyy olla yksinkertaisia, eikä niiden ei tulisi häiritä kehitysprojektia
3. Jotta turvallisuuden suunnittelun menetelmä voitaisiin onnistuneesti integroida ketterän kehittämisen menetelmiin, tulisi sen tarjota konkreettista tukea ja tarvittavat työkalut kehitystyön jokaisessa vaiheessa, vaatimusmäärittelystä testaukseen

4. Onnistuneen turvallisuuskomponentin tulisi mukautua nopeisiin ja jatkuviin muutoksiin sekä tukea useiden inkrementaalisten iteraatioiden käsittelyä

Siponen ym. (2005) kuvaavat vaatimuksiin perustuen yleiskäyttöistä ja muokattavaa, ketterään kehitykseen sopivaa prosessia, joka sisältää turvallisuuden suunnittelun keskeiset elementit. Näitä elementtejä ovat *turvallisuuskeskeiset henkilöt* (engl. security-relevant subjects), *turvallisuuskeskeiset kohteet* (engl. security-relevant objects), *henkilöiden ja kohteiden turvallisuusluokittelu* (engl. security classification of objects and subjects) sekä *riskinhallinta* (engl. risk management). Elementtejä sovelletaan kehityksen eri vaiheissa eli vaatimusmäärittelyssä, suunnittelussa, toteutuksessa ja testauksessa. Vaiheet eivät ole välttämättä peräkkäisiä, ja jokainen niistä on valinnainen. (Siponen ym., 2005.)

Vaatimusmäärittelyssä on neljä osa-aluetta, jotka ovat turvallisuuskeskeisten kohteiden ja henkilöiden tunnistaminen vaatimusmäärittelystä, niiden luokittelu, sekä riskien analysointi ja niiden hallinta. Kehittäjät tunnistavat asiakkaan kanssa käytävien keskusteluiden perusteella organisaation olennaiset suojattavat kohteet, jotka ovat mahdollisia turvallisuuskeskeisiä kohteita. Kohteet ilmaistaan ketterän kehittämisen notaatiolla, kuten turvallisuudella täydennetyillä käyttötapauksilla (Siponen & Baskerville, 2001), ja notaatiota käytetään kohteiden dokumentaationa. (Siponen ym., 2005.)

Turvallisuuskeskeisten kohteiden lisäksi vaatimusmäärittelyssä on tunnistettava henkilöt tai ryhmät, joilla on pääsy järjestelmään ja jotka voivat vahingossa tai tahallisesti vaarantaa järjestelmän turvallisuuden. Edelleen asiakaskeskusteluiden perusteella voidaan tunnistaa nämä mahdolliset turvallisuuskeskeiset henkilöt, ja turvallisuuskohteiden tavoin dokumentointi tehdään esimerkiksi turvallisuuden käyttötapauksilla. (Siponen ym., 2005.)

Kehittäjien tulee myös tunnistaa turvallisuuskohteista tärkeimmät ja arkaluontoisimmat kohteet. Tämä tehdään niiden luokittelulla, jossa käytetään esimerkiksi luottamuksellisuusluokkia. Tämän jälkeen analysoidaan, kenellä tulisi olla pääsyoikeus kyseisiin kohteisiin, ja myös tässä vaiheessa käyttötapaukset ovat hyödyllinen apuväline. Kehittäjät voivat varmistua turvallisuuskeskeisten kohteiden ja henkilöiden oikeasta luokittelusta laatimalla ehdokkaat käyttötapauksista luokittelun yhteydessä. (Siponen ym., 2005.)

Vaatimusmäärittelyn yhtenä osana on myös riskien analysointi ja niiden hallinta. Turvallisuuskeskeisten kohteiden ja henkilöiden tunnistamisen sekä niiden luokittelun jälkeen kehittäjien on analysoitava mahdolliset uhkat, ja tässä voidaan käyttää apuna esimerkiksi alaluvussa 2.4 esiteltyjä McDermottin ja Foxin (1999) väärinkäyttötapauksia. Siponen ym. (2005) lisäävät myös arvioidun kustannuksen tapahtuneesta hyökkäyksestä toipumiseen, mikä mahdollistaa yksinkertaisen riskianalyysin ja -hallinnan.

Suunnitteluvaiheessa varmistetaan, että turvallisuusvaatimukset ovat mukana suunnitelmissa. Turvallisuuskeskeiset kohteet ja henkilöt sisällytetään ketterän kehityksen suunnittelumalleihin siten, että niissä on mukana turvalli-

suusluokittelu. Tarvittaessa suunnittelijat käyttävät riskienhallinnan tekniikoita toimintojen priorisoinnissa. (Siponen ym., 2005.)

Toteutusvaiheessa turvallisuusvaatimukset ja tunnistettujen uhkien vastatoimet tulee toteuttaa niiden arkaluonteisuuden mukaan. Toteutus tehdään prioriteettijärjestyksen mukaan siten, että korkeamman prioriteetin toiminnot toteutetaan ensin. (Siponen ym., 2005.)

Testauksella varmistetaan toteutettujen ominaisuuksien toiminnallisuus ja kehittäjät hyödyntävät testauksessa niin käyttötapauksia kuin väärinkäyttötapauksia. Kuten toteutuksessa, myös testauksessa tulee olla priorisoitu lista testitapauksista ja korkeamman prioriteetin toiminnallisuus testataan ensin. (Siponen ym., 2005.)

Siponen ym. (2005) soveltavat edellä mainittuja turvallisuuden suunnittelun elementtejä esimerkkinä FDD :n (Palmer & Felsing, 2001) ketterään menetelmään. Sovellusta ei kuitenkaan käsitellä tässä tutkielmassa tarkemmin, sillä Siposen ym. (2005) esittämiä käytänteitä voidaan soveltaa myös muissa ketterissä menetelmissä ja kuvatut turvallisuuden suunnittelun elementit mahdollistavat tutkimusongelman kannalta keskeisten kysymysten pohdinnan.

4.3.5 Turvallinen ja ketterä Web-kehitys

Kongsli (2006) lisää turvallisuuden käytänteitä ketterään kehitykseen ja tarkastelee turvallisuuden integrointia erityisesti Web-sovellusten kehityksen näkökulmasta. Hän ei esitä käytettäväksi tiettyä prosessimallia, vaan esittelee ketterän lähestymistavan periaatteiden mukaiset turvallisuuden suunnittelun käytänteet, joita on testattu käytännön kehitystyössä. Esitettyjä käytänteitä ovat väärinkäytön kertomukset, automaattiset yksikkö- ja hyväksyntätestit, läpäisytestit sekä järjestelmän koventaminen ja turvallinen käyttöönotto.

Väärinkäytön kertomukset (engl. misuse stories) ovat Kongslin (2006) mukaan yksi variaatio aiemmin esitetyistä väärinkäytön kertomuksista. Hän mainitsee, että tavalliset käyttäjäkertomukset ovat osoittautuneet tehokkaaksi työkaluksi toiminnallisten vaatimusten ilmaisemiseen, joten väärinkäytön kertomusten tulisi toimia vastaavasti turvallisuusvaatimuksissa. Käytänteenä väärinkäytön kertomukset laaditaan siten, että meneillään olevaan iteraatioon valituista käyttäjäkertomuksista mallinnetaan siihen liittyvä väärinkäytön kertomus. Keskeisenä kysymyksenä on: *"Kuinka tätä toiminnallisuutta voidaan käyttää väärin?"* (Kongsli, 2006, s.2). On huomionarvoista, että yhteen käyttäjäkertomukseen voi liittyä yksi, monta tai ei yhtään väärinkäytön kertomusta. Toisaalta väärinkäytön kertomuksella ei välttämättä ole siihen liittyvää käyttäjäkertomusta. Kaikkia turvallisuusnäkökohtia ei voida näin ollen huomioida ainoastaan johtamalla väärinkäytön kertomuksia käyttäjäkertomuksista, ja nämä väärinkäytön kertomukset on mallinnettava erikseen. (Kongsli, 2006.)

Väärinkäytön kertomuksen laatimisen jälkeen on mietittävä, kuinka sen toteutuminen voidaan estää. Tavallisissa käyttäjäkertomuksissa voidaan luoda yksinkertaisesti hyväksyntätestaus, mutta väärinkäytön kertomuksissa on Kongslin (2006) mukaan useita eri tapoja estää sen toteutuminen. Hän mainit-

see ensimmäiseksi askeleeksi väärinkäytön kertomusten käsittelyssä turvallisuuspuutteen lähteen tunnistamisen. Tällä tarkoitetaan joko arkkitehtuuriin, toteutukseen tai näistä molempiin liittyvää ongelmaa. Monimutkaisemmissa väärinkäytön kertomuksissa käytetään apuna *hyökkäyspuita* (Amoroso, 1994), joiden avulla voi löytyä useita mahdollisia haavoittuvuuksia. (Kongsli, 2006.) Hyökkäyspuiden Amoroson (1994) malli on vastaava kuin alaluvussa 2.5 esitetyt hyökkäyspuut Schneierin (1999) mukaan.

Ketterien periaatteiden mukaisesti väärinkäytön kertomus tulisi olla testattavissa hyväksyntätestauksella ja ideaalitilanteessa asiakas itse tekee hyväksyntätestin. Jos väärinkäytön kertomuksen eri variaatiot voidaan kattaa riittävällä tasolla, on hyväksyntätesti samalla vaatimusmäärittely toteutettavalle turvallisuustoiminnolle. Useissa tilanteissa hyväksyntätestausta ei kuitenkaan voida luoda väärinkäyttötapauksille, sillä tyypillisesti testit toteutetaan käyttäjän näkökulmasta ja järjestelmän käyttöliittymälle. Järjestelmään kohdistuvat hyökkäykset tapahtuvat usein käyttöliittymän ulkopuolelta esimerkiksi tiedostojärjestelmään tai tietokantaan, jolloin tarvitaan hyökkäystä kuvaavia yksikkötestejä. Nämä testit keskittyvät paremmin järjestelmän sisäänpääsyn pisteisiin. Kongsli (2006) mainitsee automaattiset testit yhdeksi tärkeimmistä ketterän kehittämisen käytänteistä, ja hänen mukaansa ne ovat samalla tavalla hyödyllisiä myös turvallisuuden toiminnallisuuksissa. Automaattinen testaus mahdollistaa ongelmiin puuttumisen välittömästi niiden ilmaantumisen jälkeen. Yksikkötestauksilla järjestelmää voidaan jatkuvasti refaktoroida, ja hyväksyntätestit toimivat muodollisena järjestelmän vaatimusmäärittelynä. (Kongsli, 2006.)

Kongsli (2006) esittelee uuden kokouskäytänteen turvalliseen ketterään kehittämiseen. *Turvallisuuskatselmoinnissa* (engl. security review meeting) tarkastellaan väärinkäytön kertomuksia, mikäli niitä on paljon, ne ovat monimutkaisia tai ne vaativat suurempia muutoksia järjestelmään. Kongsli (2006) mainitsee katselmointiin liittyen esimerkkinä Scrumin (Schwaber & Beedle, 2002; Schwaber & Sutherland 2013), jossa turvallisuuskatselmointi pidetään iteraation suunnittelutapaamisen jälkeen. Kehitystiimi osallistuu tapaamiseen, ja sillä varmistetaan, että koko tiimi keskittyy yhteisvastuullisesti turvallisuuteen (Kongsli, 2006).

Ketterässä lähestymistavassa iteraatioiden tulisi tuottaa toimiva tuote. Tämän vuoksi vähintään testausympäristö tulisi olla käytössä projektin alkuvaiheista lähtien. Turvallisuuden kannalta tämä tarkoittaa järjestelmän kokoonpanon suojaamista jo varhaisessa vaiheessa kehitystyötä. Kongslin (2006) käytännön projekteista saatujen kokemusten perusteella esimerkiksi järjestelmän läpäisytestaukset on voitu aloittaa ensimmäisistä iteraatioista alkaen. Hän kuitenkin huomauttaa, että läpäisytestejä on suoritettava useita kertoja ja kaikissa ympäristöissä projektin loppuun saakka. Perusajatuksena on joka tapauksessa turvallisuuden huomioiminen mahdollisimman varhaisessa vaiheessa ketterien periaatteiden mukaisesti. (Kongsli, 2006.)

Kongslin (2006) mukaan esiteltyjen käytänteiden hyötyjä ovat lisääntynyt turvallisuustietoisuus kehitystiimissä, turvallisuuden yhteisomistajuus, turvallisuuden suunnittelun yksinkertaistuminen, automaattisten testien hyödyntä-

minen, sekä luopuminen ”läpäise ja testaa” – periaatteesta. Mahdollisiksi puutteiksi mainitaan väärinkäytön kertomusten ja testien puutteellisuus sekä mahdollinen tarve turvallisuusasiantuntijalle. Kongsli (2006) mainitseekin, että esitetyt käytänteet edesauttavat turvallisuutta ketterässä kehityksessä, mutta eivät ole vielä itsessään riittäviä.

4.4 Esityksiä XP:n ja Scrumin räätälöinneistä

Seuraavaksi käsitellään XP:n ja Scrumin räätälöintejä, joissa turvallisuuden suunnittelu on huomioitu osana ketterää kehittämistä. Esityksistä käsitellään niiden käytänteet, roolit ja esitettävät muutokset kehittämisprosessiin. Kaksi ensimmäistä kohtaa käsittelevät XP:hen perustuvia räätälöintejä, ja kaksi viimeistä Scrumin räätälöintejä.

4.4.1 Extreme Security Engineering (XSE)

Beznosov (2003) yhdistää ketterän kehittämisen ja erityisesti XP:n käytänteitä turvallisuuden suunnittelun käytänteisiin *Extreme Security Engineering* (XSE) – menetelmällä. XSE:n tarkoituksena on mukautua ketterän kehittämisen tavoin kehitystyössä tapahtuviin muutoksiin esimerkiksi vaatimusten ja käytettävien teknologioiden suhteen. Sen perusperiaatteena on saavuttaa *riittävä turvallisuustaso* (engl. good enough security) määrittelemättä etukäteen mikä se on. XSE:ssä kuvataan käytännössä joukko XP:n käytänteitä turvallisuuden suunnittelun näkökulmasta. Nämä käytänteet on esitetty seuraavissa kappaleissa.

Suunnittelupeli toteutetaan kuten XP:ssä ja sen tarkoituksena on suunnitella pieniä julkaisuja lyhyillä iteraatioilla siten, että riittävä turvallisuustaso voidaan saavuttaa (Beznosov, 2003). XP:n suunnittelupelissä kehittäjät arvioivat käyttäjäkertomusten perusteella toteutuksen vaatiman työmäärän ja asiakas päättää julkaisujen sisällön sekä ajoituksen (Beck, 1999a). Tämä vastaa XSE:n suunnittelupeliä, mutta Beznosov (2003) lisää kuitenkin suunnittelupelin käyttäjäkertomuksiin turvallisuusratkaisut ja niiden testauksen.

Käyttäjäkertomukset mahdollistavat turvallisuusvaatimusten suunnittelun vapaamuotoisella kielellä, jolloin asiakkaat voivat määritellä, mitä he haluavat nähdä turvallisuusratkaisujen toteutuksessa. Tämä parantaa Beznosovin (2003) mukaan osaltaan turvallisuuskriittisten projektien onnistumista, mutta keskeistä on kuitenkin XP:n muiden käytänteiden yhdistäminen käyttäjäkertomuksiin. Beznosov (2003) mainitsee näistä käytänteistä tärkeimmäksi pienet julkaisut lyhyillä iteraatioilla sekä testauksen. Pienillä julkaisuilla käyttäjäkertomuksia voidaan päivittää ja priorisoida uudelleen riittävän usein, jolloin riittävä turvallisuustaso voi vaihdella teknologian ja ympäristön mukaan. Testauksella puolestaan voidaan varmistaa se, että toteutetut turvallisuusratkaisut tukevat kaikkia käyttäjäkertomuksia. (Beznosov, 2003.)

Pienet julkaisut voivat olla Beznosovin (2003) mukaan vaikea toteuttaa turvallisuuskriittisissä projekteissa. Pienet julkaisut muodostavat kuitenkin perustan, joka tukee muita XP:n käytänteitä, ja niillä voidaan saavuttaa muun muassa aikainen palaute, toiminnallisuus varhaisessa vaiheessa, muutosten toteutettavuus sekä edellisten iteraatioiden hyödyntäminen tulevien turvallisuusratkaisujen kehittämisessä. Tämän vuoksi myös turvallisuusratkaisut tulisi pilkkoa pienempiin osiin, sillä saatava hyöty korvaa pilkkomisesta aiheutuvan ylimääräisen työmäärän. Käytänteenä pilkkominen tapahtuu vastaavasti kuin ohjelmistoprojekteissa, sillä useimmissa turvallisuuskriittisissä projekteissa voidaan tunnistaa pieniä turvallisuuden toiminnallisuuksia, jotka voidaan julkaista jo projektin alkuvaiheessa. (Beznosov, 2003.)

Käyttäjäkertomukset eivät yksinään voi toimia turvallisuusvaatimuksina. Kertomukset liitetään yhteen tai useampaan testitapaukseen ja *testauksella* voidaan varmistaa, että suunnitellut toiminnot on toteutettu ja ne toimivat oikein. Toiminnallisen testauksen lisäksi tehdään yksikkötestejä, jotka voivat olla turvallisuuskriittisissä projekteissa normaaleja ohjelmistoprojekteja vaikeampi toteuttaa muun muassa käytettävien teknologioiden suuren määrän vuoksi. Turvallisuustoiminnot voivat myös usein olla vaikeita testata niiden negatiivisen luonteen vuoksi, mikä asettaa osaltaan haasteita testien toteuttamiselle. (Beznosonv, 2003.) XSE:n testaus liittyy keskeisesti turvallisuuden varmistamiseen, joka voidaan nähdä Farrohan ja Farrohan (2011) mukaan luottamuksen tasona siitä, että ohjelmiston toiminnallisuudet ovat tarkoituksenmukaisia ja ohjelmistossa ei ole tarkoituksella tai vahingossa suunniteltuja haavoittuvuuksia.

Beznosovin (2003) mukaan integrointi voi olla suuri osa turvallisuuslähtöisten projektien työmäärästä ja myös XP:n *jatkuvan integroinnin* käytänteen soveltaminen XSE:ssä on hyödyllistä. Erityisesti käyttöönoton yhteydessä jatkuva integrointi tuo lisäarvoa projektin toiminnalle. Se auttaa myös turvallisuuden suunnittelijoita ymmärtämään asiakasympäristön toimintaa etenkin XP:n testaukseen ja lyhyisiin iteraatioihin yhdistettynä. Ongelmana jatkuvan integroinnin soveltamisessa on kuitenkin alkuvaiheen ympäristön kehittäminen, sillä kustannukset voivat nousta korkeiksi ja ympäristö voi olla myös vaikea toteuttaa. (Beznosov, 2003.)

Yksinkertainen suunnittelu ja refaktorointi ovat keskeisiä käytänteitä turvallisuuslähtöisissä projekteissa varhaisen ja jatkuvan toimituksen sekä odottamattomien muutosten kannalta. Yksinkertainen suunnittelu voi kuitenkin olla vaikeaa projekteissa, jotka sisältävät paljon COTS-tuotteita tai -komponentteja, jotka eivät ole ohjelmistopohjaisia. (Beznosov, 2003.)

Beznosov (2003) kokee ketterän kehittämisen aineistoon viitaten *pariohjelmoinnin* lisäävän kehitettävien tuotteiden laatua myös turvallisuuslähtöisissä projekteissa, vaikkakin hän mainitsee, että oletus ei varsinaisesti perustu mihinkään aineistoon. Näkemys on osittain ristiriitainen Dybån ja Dingsøyrin (2008) kanssa, joiden mukaan pariohjelmointia ei välttämättä koeta hyödylliseksi käytännön kehitystyössä. Toisaalta Beznosovin (2003) mukaan pariohjelmoinnilla voidaan saavuttaa laadukkaampi lopputulos tehokkuuden kustan-

nuksella, mikä selittänee osaltaan ristiriitaisia näkemyksiä käytänteen hyödyllisyydestä.

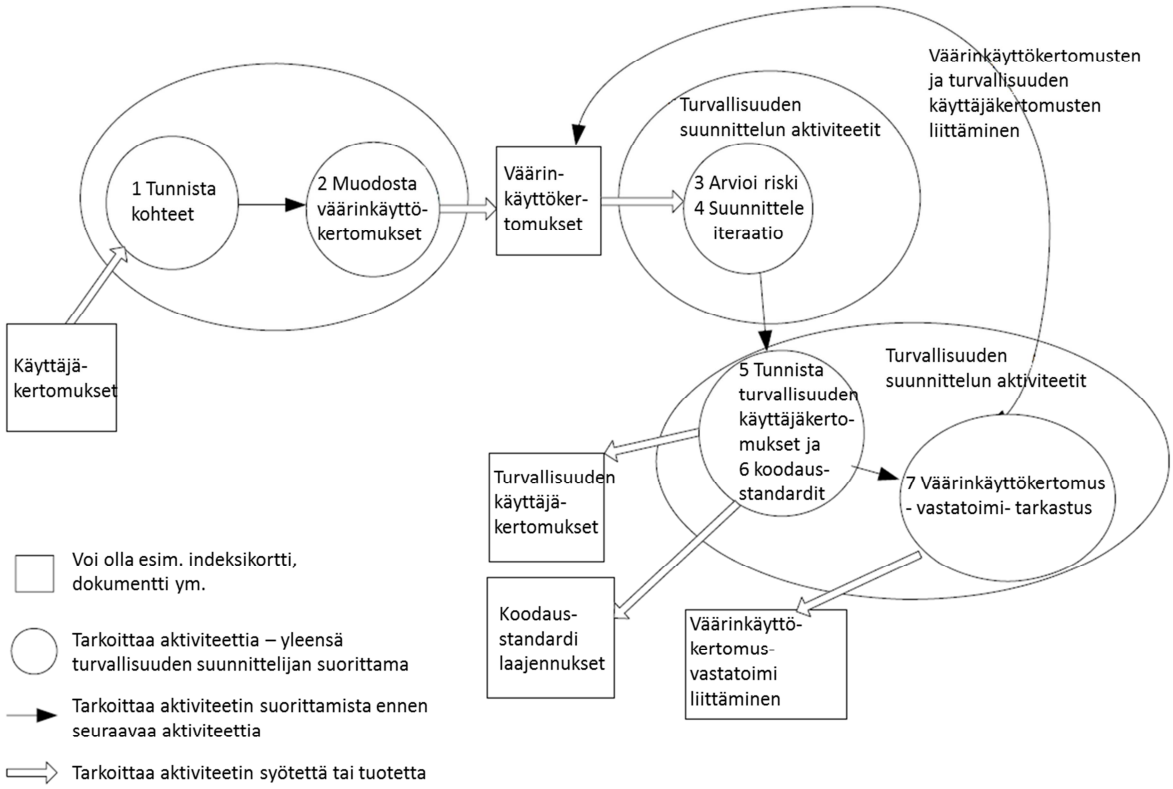
Asiakkaan läsnäolo toteutuu XSE :ssä siten, että asiakas vastaa muun muassa käyttäjäkertomusten kirjoittamisesta, kertomusten jakamisesta julkaisuihin, sekä niiden uusien versioiden kirjoittamisesta. Asiakas toimittaa myös turvallisuusasiantuntijoille turvallisuuden suunnitteluun tarvittavat yksityiskohdat. (Beznosov, 2003.) Käytänne on yhteneväinen XP :n kanssa, kuitenkin siten, että asiakkaalla on rooli järjestelmän toiminnallisuuden kehittämisen lisäksi myös turvallisuuden suunnittelussa.

4.4.2 XP:n laajennus turvallisuuden vaatimusmäärittelyyn

Boström ym. (2006) esittävät oman turvallisuuspainotteisen version XP:stä, jossa laajennetaan XP:n käytänteitä tukemaan paremmin turvallisuuden vaatimusmäärittelyä ja suunnittelua. Menetelmä keskittyy XP:n suunnittelupelin laajentamiseen, sillä normaali suunnittelupeli ei Boströmin ym. (2006) mukaan riitä turvallisuusvaatimusten määrittelyyn. Käyttäjäkertomuksia täydennetään Peetersin (2005) kaltaisilla *väärinkäyttökertomuksilla* (engl. abuser stories) ja *turvallisuuden käyttäjäkertomuksilla* (engl. security-related user stories). Tärkeää suunnittelupelin laajentamisessa on yhteensopivuus XP:n käytänteisiin ja iteratiiviseen kehitykseen. Vaatimusmäärittelyprosessin tuotteet tulisi olla helposti jäljitettävissä ja sovitettavissa ulkoiselle katselmoinnille. Vaatimusten keräämisen tulisi myös tukea ennakoivasti turvallisuusvaatimuksia ja vaatimusten tulisi perustua riskianalyysiin. (Boström ym., 2006.) Näiden periaatteiden pohjalta XP:n suunnittelupeliin esitetään joukko aktiviteetteja ja käytänteitä (kuvio 9), jotka käsitellään tarkemmin seuraavissa kappaleissa. Numeroilla viitataan vastaaviin kohtiin kuviossa.

Suojattavat *kohteet tunnistetaan* XP:n normaalien käyttäjätarinoiden laatimisen yhteydessä (kohta 1). Tavoitteena on saavuttaa ymmärrys kehitettävän järjestelmän sellaisista kohteista, joilla on arvoa organisaatiolle tai järjestelmän käyttäjille. Työskentely toteutetaan turvallisuuden suunnittelijan johdolla XP-tiimin ja asiakkaan yhteistyönä. Tunnistetut kohteet dokumentoidaan suunnittelutaululle. (Boström ym., 2006.)

Väärinkäyttökertomukset laaditaan suojattavien kohteiden tunnistamisen kanssa rinnakkain (kohta 2). Kertomukset ovat skenaarioita uhkista, jotka voivat muodostaa kasvaneen riskin tunnistetuille kohteille, ja ne laaditaan vapaamuotoisena tekstinä indeksikorteille kuten tavalliset käyttäjäkertomukset. Turvallisuuden suunnittelija laatii väärinkäyttökertomukset, mutta siihen voivat osallistua myös kehittäjät ja asiakas. (Boström ym., 2006.) Periaate on sama kuin Singhalin ja Singhalin (2011) sekä Peetersin (2005) esityksissä, joiden mukaan väärinkäyttökertomukset tuotetaan asiakkaan ja kehittäjien yhteistyönä.



KUVIO 9 Yleiskuvaus laajennetusta XP:n suunnittelupelistä (pohjautuen Boström ym., 2006, s.3)

Riskien arviointi aloitetaan väärinkäyttökertomusten ja käyttäjäkertomusten laatimisen jälkeen (kohta 3). Arvioinnin perustana ovat väärinkäyttökertomuksissa tunnistetut uhkat, ja vaihe suoritetaan turvallisuuden suunnittelijan ja asiakkaan yhteistyönä. Turvallisuuden suunnittelija vastaa riskien arvioinnista, mutta asiakkaan osallistuminen on kuitenkin keskeistä kohdealueen hyvän tuntemuksen vuoksi. (Boström ym., 2006.)

Riskien arvioinnin jälkeen aloitetaan *iteraation suunnittelu* (kohta 4) vastaavasti kuin tavallisessa XP:n suunnittelupelissä. Asiakas päättää turvallisuuden suunnittelijan ohjauksella, mitkä väärinkäyttökertomukset toteutetaan iteraation aikana. Ensimmäisenä otetaan huomioon korkeimman riskin omaavat kertomukset. (Boström ym., 2006.)

Seuraavaksi tunnistetaan turvallisuuden käyttäjäkertomukset, jotka täydentävät väärinkäyttökertomuksia (kohta 5). Kertomuksilla on selkeä käyttötarkoitus, sillä väärinkäyttökertomukset keskittyvät uhkien tunnistamiseen, mikä ei yksin riitä määrittelemään tarvittavaa vastatoimea uhkalle. Turvallisuuden käyttäjäkertomuksissa näiden vastatoimien toteuttaminen on keskeistä. Käytännössä ne ovat toiminnallisia turvallisuusvaatimuksia, joilla väärinkäyttökertomuksissa tunnistettujen uhkien vastatoimet voidaan toteuttaa (Boström ym., 2006). Turvallisuuden kertomukset laaditaan turvallisuuden suunnittelijan, asiakkaan ja kehittäjien yhteistyönä. Samalla aktiviteetti toimii määrittelyvai-

heena järjestelmän turvallisuusarkkitehtuurille, ja turvallisuuden käyttäjäkertomukset ovat myös testattavia järjestelmän osia. (Boström ym., 2006.)

Kaikkia väärinkäyttökertomusten uhkia ei voida Boströmin ym. (2006) mukaan ottaa huomioon turvallisuuden käyttäjäkertomuksilla, vaan osa tärkeistä uhkista tulee huomioida koko järjestelmän kattavilla ominaisuuksilla. *Turvallisuuden koodausstandardit* (engl. security-related coding standards) on tarkoitettu tällaisten koko järjestelmän kattavien ominaisuuksien toteuttamiseen (kohta 6). Esimerkkejä koodausstandardeista ovat *turvallisen koodauksen tekniikat* (engl. secure coding techniques) ja *suunnittelun säännöt* (engl. design rules). Boström ym. (2006.)

Turvallisuuden käyttäjäkertomuksella täytyy olla jokin siihen liittyvä väärinkäyttökertomus, mikä varmistetaan *väärinkäyttökertomusten ja vastatoimien tarkastuksella* (engl. abuser story – countermeasure cross-checking) (kohta 7). Kuten Boström ym. (2006) mainitsevat, aktiviteetti on hieman päällekkäinen aiempien aktiviteettien kanssa, sillä turvallisuuden käyttäjäkertomukset johdetaan väärinkäyttökertomuksista. Heidän mukaansa kertomusten liittäminen toisiinsa on kuitenkin tärkeää ja uhkien vastatoimet tulee olla oikein, joten tarkastus on tämän vuoksi perusteltua. Turvallisuuden suunnittelija vastaa tarkastuksesta, mutta epäselvissä tilanteissa asiakas tarkentaa tarvittaessa käyttäjäkertomuksia (Boström ym., 2006).

4.4.3 Scrum ja turvallisuuden kehitysjono

Azham, Ghani ja Ithnin (2011) ovat räätälöineet Scrumista paremmin turvallisuuden suunnitteluun sopivan version. Keskeistä menetelmässä on *turvallisuuden kehitysjono* (engl. security backlog), jonka avulla turvallisuus voidaan huomioida paremmin osana järjestelmän ketterää kehitystä. Kehitysjonon lisäksi Scrumia esitetään laajennettavaksi uudella roolilla, *turvallisuusvastaavalla* (engl. security master), joka johtaa turvallisuuden suunnittelua. Seuraavissa kappaleissa tarkastellaan näitä Scrumin laajennuksia ja uusia käytänteitä.

Azham ym. (2011) toteavat, että turvallisuusaktiviteettien lisääminen osaksi Scrumin tavallista tuotteen kehitysjonoa ei riitä turvallisuuden suunnitteluun. Heidän mukaansa asiakkaalla on yleensä huono tuntemus mahdollisista turvallisuusriskeistä, joten turvallisuuden suunnitteluun tarvitaan erillinen kehitysjono. Näin voidaan tunnistaa järjestelmän haavoittuvuuksia ja hallita turvallisuusriskejä, säilyttämällä kuitenkin kehitystyön ketteryys. Turvallisuuden kehitysjonon hallinnassa käytetään turvallisuuden suunnittelun periaatteita, joiksi mainitaan Azhamin ym. (2011, s.1 ja s.2) mukaan seuraavat:

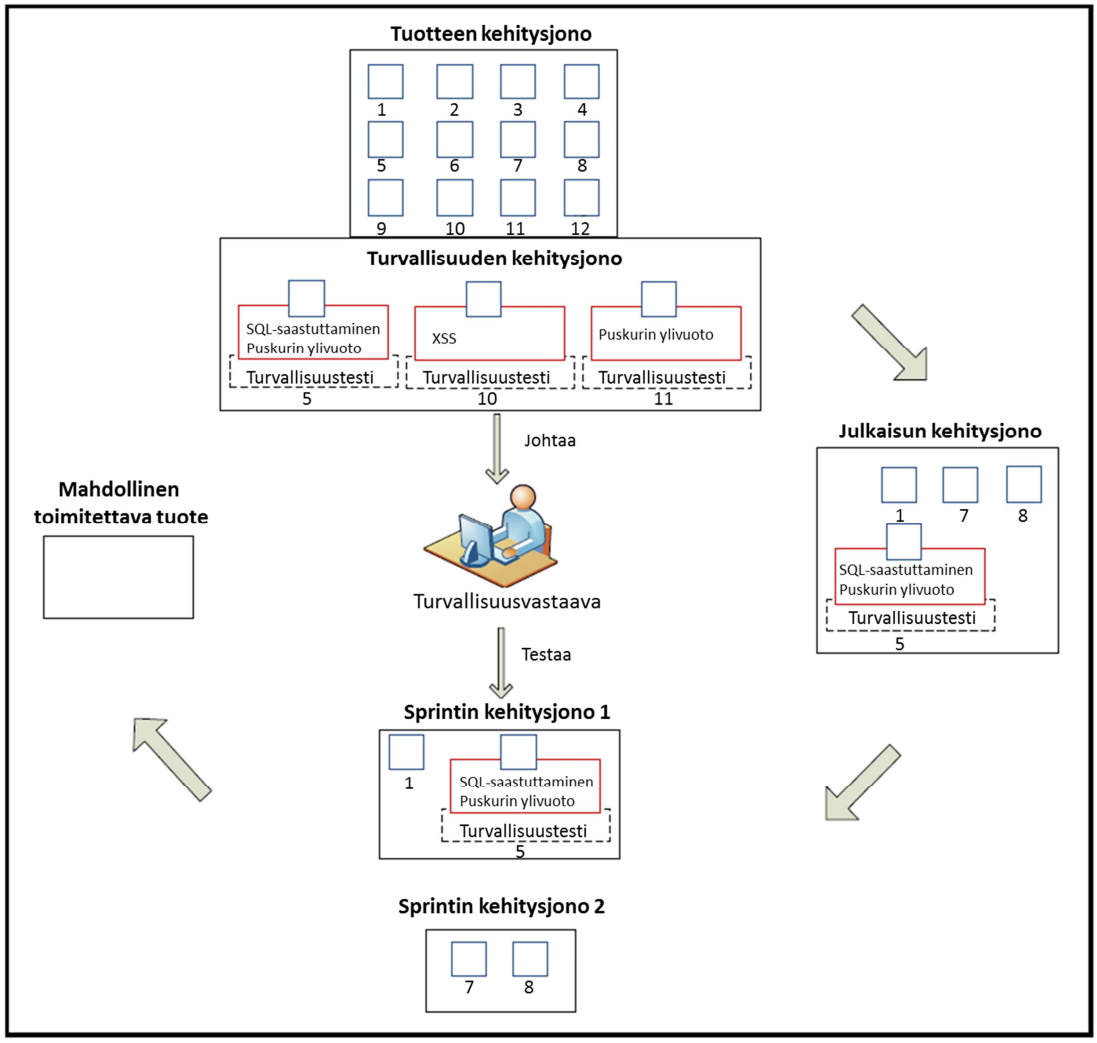
- *Minimioikeuksien periaate* (engl. the principle of least priviledge): kohteeseen annetaan vain pienimmät mahdolliset käyttöoikeudet
- *Turvallisen virheen periaate* (engl. the principle of failing securely): jos järjestelmän suoritus päättyy virheeseen, sen tulee tapahtua turvallisesti

- *Heikoimman lenkin turvaamisen periaate* (engl. the principle of securing the weakest link): hyökkäykset ovat todennäköisimpiä järjestelmän vähiten suojattuihin osiin
- *Suojauksen syvyyden periaate* (engl. the principle of defence in depth): järjestelmän suojausmekanismien eri kerroksilla voidaan pienentää onnistuneen hyökkäyksen todennäköisyyttä
- *Oikeuksien erottamisen periaate* (engl. the principle of separation of priviledge): ennen kuin oikeus kohteeseen myönnetään, tulee useiden ehtojen täyttyä
- *Mekanismien taloudellisuuden periaate* (engl. the principle of economy of mechanism): järjestelmän suunnittelu tulee pitää yksinkertaisena
- *Vähiten yleisen mekanismin periaate* (engl. the principle of least common mechanism): eri kohteiden oikeuksien myöntämisessä tulisi välttää saman turvallisuusmekanismin käyttöä
- *Täyden välityksen periaate* (engl. the principle of complete mediation): järjestelmän on varmistettava, että pääsynhallintaa pyytävä kohde on oikea

Turvallisuusvastaavan tehtävänä on tunnistaa turvallisuuden periaatteiden mukaisesti järjestelmän toiminnot, joissa on mahdollinen turvallisuusriski. Turvallisuusvastaava dokumentoi riskit turvallisuuden kehitysjonoon ja tekee valituille toiminnoille turvallisuustestauksen. (Azham ym., 2011.)

Vaikka turvallisuuden kehitysjono onkin erillinen muista Scrumin kehitysjonoista, on se keskeisesti sidoksissa Scrum-viitekehyksessä käytettäviin tavallisiin kehitysjonoihin (ks. kuvio 10). Turvallisuusvastaava käy läpi kaikki tuotteen kehitysjonon kohteet ja poimii niistä osan turvallisuuden kehitysjonoon, johon toimintojen turvallisuusvaatimukset dokumentoidaan kehittäjiä ja testausta varten (Azham ym., 2011). Valitut kohteet siirretään tarkastelun jälkeen turvallisuuden kehitysjonosta tavalliseen sprintin kehitysjonoon ja edelleen julkaisun kehitysjonoon. Kohteet erotetaan tavallisista kehitysjonon kohteista punaisella- ja katkoviivakehyksellä, mikä auttaa muun muassa tiimin ja asiakkaiden yhteistyötä turvallisuuden toiminnallisuuksista keskusteltaessa. (Azham ym., 2011.)

On huomioitavaa, että kohteet Scrum-kehitysjonoissa eivät siis muutu, mutta niitä täydennetään turvallisuusvaatimuksilla. Näin erot tavalliseen kehitysprosessiin ja Scrum-viitekehyksen käyttämiseen ovat vähäiset. Toisaalta Scrumin täydentäminen erillisellä turvallisuuden kehitysjonolla takaa Azhamin ym. (2011) mukaan yksinkertaisen, mutta riittävän dokumentoinnin turvallisuusriskien huomioon ottamisesta. On myös tärkeää, että turvallisuus huomioidaan osana kehitystyötä koko kehitysprosessin ajan (Farroha & Farroha, 2011; Singhal & Singhal, 2011). Koska turvallisuusvastaava käy läpi jokaisen Scrum-työlistan kohteen, on turvallisuus huomioitu suunnittelusta julkaisuun saakka ja turvallisuusmekanismit voidaan kehittää lisäämättä niitä järjestelmään jälkikäteen.



KUVIO 10 Scrum-prosessin laajennus turvallisuuden kehitysjonolla (Azham ym., 2011, s.3)

Esitettyjen Scrum-laajennusten lisäksi Azham ym. (2011) pitävät turvallisuuskoulutusta kehittäjille ja sidosryhmille tärkeänä, sillä turvallisuuden kehitysjonossa tarvitaan taitoja turvallisuusriskien tunnistamiseen ja tarvittaviin vastatoimiin. Koulutus esitetään toteutettavaksi Gen ym. (2007) mukaisesti, jossa koulutuksen luonne vaihtelee roolin mukaan. Kehittäjät tarvitsevat enemmän turvallisuusmekanismeihin liittyvää teknistä koulutusta, kun taas muiden sidosryhmien, kuten asiakkaiden, tulee olla tietoisia esimerkiksi organisaation turvallisuuskäytänteistä tai järjestelmän turvallisesta käytöstä.

4.4.4 S-Scrum

Mougouein, Sanin ja Almasin (2013) turvallisen Scrum-viitekehyksen versio, S-Scrum, on kehitetty vastaamaan paremmin nykypäivän turvallisuusvaatimuksiin erityisesti Web-sovellusten kehityksessä. Mougouein ym. (2013) mukaan turvallisuus on yksi tärkeimmistä kysymyksistä Web-kehityksessä ja turvallisuusanalyysi ja suunnittelu on sisällytettävä osaksi kehityksen elinkaarta. Hei-

dän mukaansa Scrum ei kuitenkaan ota huomioon turvallisuutta osana järjestelmän suunnittelua, johon he esittävät Scrum-viitekehyksen räätälöintiä. S-Scrum lisää turvallisuusanalyysin ja -suunnittelun osaksi tavallista Scrum-prosessia, jolloin järjestelmän kehityksessä voidaan reagoida ketterän kehityksen periaatteiden mukaisesti muun muassa nopeisiin muutoksiin ja samalla huomioida myös turvallisuuden suunnittelun vaatimukset. Seuraavaksi esitetään Mougouein ym. (2013) laajennetun Scrum-version prosessi ja keskeiset käytänteet.

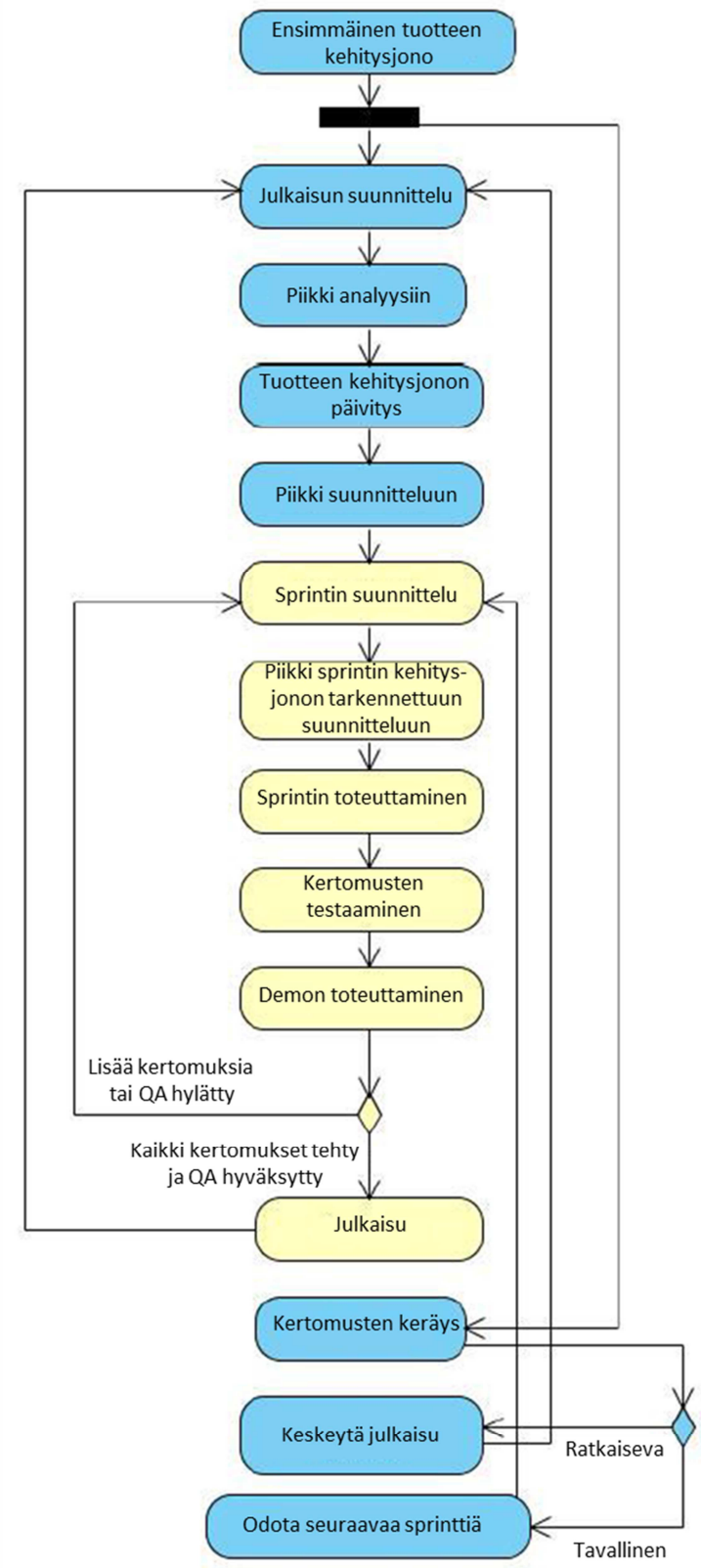
S-Scrumin prosessi vastaa tavallista Scrum-prosessia (ks. kuvio 11). Aluksi laaditaan tuotteen kehitysjono, josta prosessi jatkuu samanaikaisesti julkaisun suunnitteluun ja kertomusten keräämiseen, joilla vastataan sidosryhmien vaatimusten muutoksiin. Muutokset odottavat seuraavaan sprinttiin, ellei muutoksella ole ratkaisevaa merkitystä toteutukselle. Tällöin nykyinen julkaisu keskeytetään ja aloitetaan alusta julkaisun suunnittelulla. (Mougouei ym., 2013.)

S-Scrumin prosessissa on *piikkejä* (engl. spikes), joilla tarkoitetaan kehityksessä toteutettavia sellaisia aktiviteetteja, jotka eivät tuota asiakkaalle erityistä arvoa (Mougouei ym., 2013). Julkaisun suunnittelun jälkeen S-Scrum–prosessissa seuraa *piikki turvallisuuden analyysiin* (engl. spike for security analysis). Mougouei ym. (2013) eivät esitä tähän käytettäväksi mitään erityistä tekniikkaa, vaan mainitsevat mahdollisiksi vaihtoehdoiksi esimerkiksi väärinkäyttötapaukset (ks. alaluku 2.4) tai minkä tahansa turvallisuuden analyysiin soveltuvan menetelmän.

Turvallisuuden analyysin piikin jälkeen tuotteen kehitysjono voidaan päivittää, josta prosessi jatkuu edelleen *turvallisuuden suunnittelun piikillä* (engl. spike for design). Analyysin tulokset sovelletaan tässä vaiheessa nykyisen julkaisun malleihin. (Mougouei ym., 2013.) Kuten analyysivaiheeseen, myöskään suunnitteluun ei esitetä käytettäväksi mitään erityistä menetelmää.

Prosessi jatkuu *sprintin suunnittelulla* (engl. sprint planning). Korkean prioriteetin järjestelmän osat valitaan toteutettavaksi seuraavassa sprintissä vastaavasti kuin tavallisessa Scrum-prosessissa. Turvallisuuden analyysin yksityiskohdat tarkennetaan kuitenkin näiden valittujen kohteiden osalta *tarkennetun suunnittelun piikillä* (engl. detailed design spike). (Mougouei ym., 2013.) Edelleen tähän vaiheeseen ei esitetä tarkempia käytänteitä, vaan siinä voidaan käyttää tarkoitukseen soveltuvia turvallisuuden suunnittelun tekniikoita.

Sprintin suunnittelun ja turvallisuuden tarkennetun suunnittelun jälkeen seuraavat *toteutus* (engl. implementation), *testaus* (engl. testing) ja *demo*. Nämä vaiheet noudattavat tavallisia Scrumin käytänteitä. Jos käyttäjäkertomuksia täydennetään ja laadunvalvonta (engl. quality assurance, QA) on hyväksynyt kertomukset, seuraa *julkaisu* (engl. release). Vaihtoehtoisesti tarvitaan lisää sprinttejä, jotta käsiteltävä ongelma voidaan ratkaista. Prosessi jatkuu siihen saakka, kunnes kaikki kertomukset on toteutettu. (Mougouei ym., 2013.) Testaukseen esitetään mahdollisena tekniikkana läpäisytestausta, kuten Singhalin ja Singhalin (2011) menetelmässä. Mougouei ym. (2013) eivät kuitenkaan rajaa käytettäviä tekniikoita myöskään testauksen osalta, joten siinä on mahdollista käyttää mitä tahansa vastaavaa menetelmää.



KUVIO 11 S-Scrum prosessi (Mougouei ym., 2013, s.2)

Mougouei ym. (2013) täydentävät S-Scrumin prosessia *formaalilla mallilla* (engl. formal model), joka kuvaa prosessin eri vaiheiden tiloja ja niiden välisiä muutoksia. Malli sisältää myös säännöt eri tilojen välillä. Esimerkiksi ensimmäinen prosessin tila on analyysin piikki, joka muuttuu seuraavaan tilaan analyysin toteuttamisen jälkeen tai uuden vaatimuksen ilmetessä. Sääntöjen mukaan seuraava tila voi olla vastaavasti suunnitteluun piikki tai uusi vaatimus, josta prosessi jatkuu eteenpäin eri vaiheiden sääntöjen mukaan. Formaalia mallia ei käsitellä tarkemmin tässä tutkielmassa, sillä mallilla ei ole vaikutusta esiteltyyn prosessiin tai siihen kuuluviin käytänteisiin.

4.5 Yhteenveto

Edellä käsiteltiin turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämistä. Aluksi pohdittiin integroinnin haasteita ja hyötyjä, jossa yleisesti esiintyviksi haasteiksi todettiin eroavaisuudet turvallisuuden suunnittelun ja ketterän kehittämisen prosessissa sekä dokumentaation merkityksessä. Turvallisuuden varmistaminen nähtiin ketterän kehittämisen menetelmissä puutteellisena, ja ketterien menetelmien todettiin keskittyvän toiminnallisiin vaatimuksiin. Myös turvallisuusvaatimusten jäljittäminen ja riittävän turvallisuusosaamisen varmistaminen olivat integroinnissa todettuja haasteita. Toisaalta ketterässä kehittämisessä nähtiin turvallisuuden suunnittelua tukevia piirteitä, joiksi todettiin parantunut kommunikointi, reagointi ympäristön muutoksiin, sekä turvallisuuspuutteiden asteittainen löytäminen.

Seuraavaksi kuvattiin, miten relevantteja julkaisuja on haettu ja valittu. Työssä noudatettiin integroivan kirjallisuuskatsauksen menetelmää. Tämän jälkeen esiteltiin valittujen julkaisujen käytänteitä ja menetelmiä, joissa turvallisuuden suunnittelu ja ketterä kehittäminen on pyritty yhdistämään. Aluksi käsiteltiin väärinkäytön kertomukset, jolla turvallisuusvaatimukset voidaan kuvata ketterässä kehittämisessä käytettyjen tavallisten käyttäjäkertomusten tapaan. Seuraavaksi esiteltiin Agile Security Framework (ASF), joka on koko järjestelmän kehitysprosessin kattava ketterän turvallisuuden viitekehys. Se yhdistää eri menetelmien piirteitä, ja keskeistä viitekehyksessä on hyökkäyspuiden ja väärinkäytön kertomusten yhdistelmä, jota kutsutaan hybriditekniikaksi. Viitekehyksen jälkeen käsiteltiin kaksi ketterän kehityksen laajennusta, joista ensimmäisessä ketterää kehittämistä laajennetaan ketterän lähestymistavan kanssa yhteensopivilla turvallisuuden suunnittelun aktiviteeteilla. Aktiviteetit kattavat ketterän kehityksen vaiheet vaatimusmäärittelystä testaukseen. Toisessa esitetään yleiskäyttöistä turvallisen ketterän kehityksen prosessia, joka sisältää turvallisuuden keskeiset elementit. Näitä elementtejä ovat turvallisuuskeskeiset henkilöt, turvallisuuskeskeiset kohteet, henkilöiden ja kohteiden turvallisuusluokittelu sekä riskinhallinta. Yleisistä prosessimalleista käsiteltiin myös turvallisen ja ketterän Web-sovelluksen kehityksen prosessia, joka esittää käytänteiksi väärinkäytön kertomuksia, automaattisia yksikkö- ja hyväksyntätestejä, läpäisytestejä, sekä järjestelmän koventamista ja turvallista käyttöönottoa.

Lopuksi kuvattiin menetelmiä, joissa XP- ja Scrum-menetelmiä on laajennettu turvallisuuden suunnittelun käytänteillä. Ensimmäinen käsitellyistä räätälöinneistä oli Extreme Security Engineering (XSE), jonka periaatteena on saavuttaa riittävä turvallisuustaso määrittelemättä etukäteen, mitä se käytännössä tarkoittaa. Toisessa XP:n laajennuksessa keskeistä on suunnittelupelin laajentaminen, jolloin turvallisuuden vaatimusmäärittely ja -suunnittelu voidaan toteuttaa tavallista XP:n menetelmää tehokkaammin. Scrumin laajennuksista käsiteltiin kahta versiota, joista ensimmäisessä esitetään uudeksi käytänteeksi muun muassa turvallisuuden kehitysjono ja turvallisuusvastaavan lisääminen tiimiin. Toiseksi esiteltiin S-Scrum, joka lisää turvallisuusanalyysin ja -suunnittelun osaksi tavallista Scrum-prosessia.

5 INTEGROINTIESITYSTEN VERTAILU JA ANALYYSI

Tässä luvussa vertaillaan ja arvioidaan turvallisuuden suunnittelun ja ketterän kehittämisen integrointiesityksiä. Tarkastelun kohteena ovat edellisessä luvussa käsitellyt käytänteet ja menetelmät sekä kirjallisuuskatsauksen muu aineisto. Luvun ensimmäisessä alaluvussa kuvataan vertailussa käytetty viitekehys. Toinen alaluku sisältää menetelmien vertailua ja arviointia viitekehykseen perustuen. Luvun lopuksi esitetään keskeisestä sisällöstä yhteenveto.

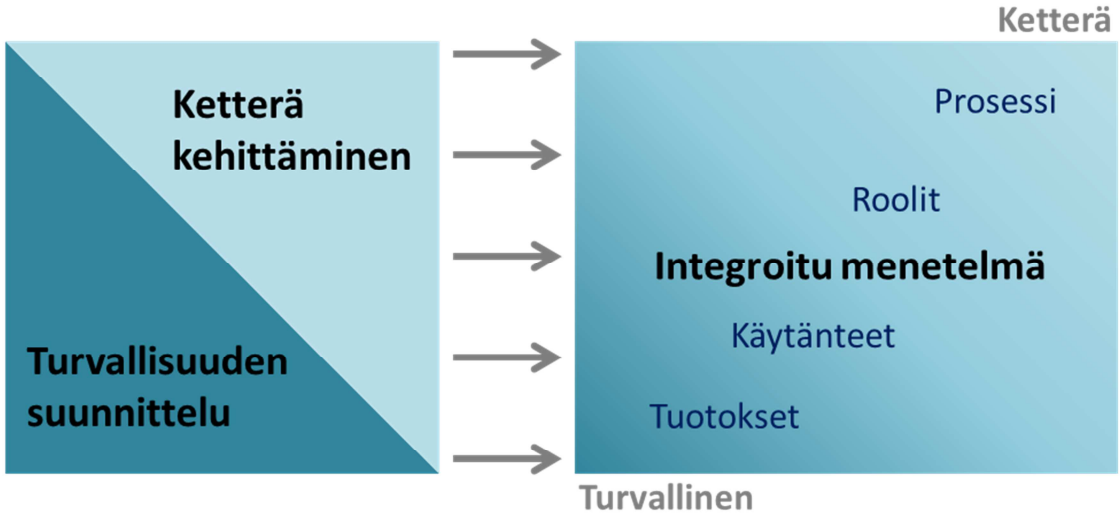
5.1 Vertailun viitekehys

Kirjallisuuskatsausta ja integrointiesitysten tarkastelua varten luotiin viitekehys (ks. kuvio 12). Viitekehyksen perustana on käytetty Jacobsonin, Boochin ja Rumbaughin (1999) mallia, jonka mukaan ohjelmistokehityksessä keskeisiä ovat *ihminen, projekti, tuote* ja *prosessi*. Ihmiset ovat tärkeä osa kehitystyötä, ja niillä on eri rooleja, mikä valittiin tämän tutkielman viitekehyksen yhdeksi osaksi. Projekteissa tehdään tuote, joka voi olla koodin lisäksi esimerkiksi suunnittelumalli tai ohjelman osa (Jacobson ym., 1999). Projekti- ja tuote-käsitteet yhdistettiin viitekehyksessä yksinkertaisesti käytänteiksi ja tuotoksiksi. Prosessilla ohjataan projekteja (Jacobson ym., 1999), ja prosessi-käsite on mukana myös viitekehyksessä hieman eri kontekstissa.

Viitekehyksen *prosessissa* tarkastellaan esitettyä kehittämisen prosessimallia tai prosessin osaa, johon integrointiesitys on tarkoitettu. Prosessi voi perustua johonkin olemassa olevaan menetelmään, kuten XP:hen tai Scrumiin. Menetelmässä voidaan myös esittää käytettäväksi uutta prosessia. Tarkastelun kohteena on prosessimallin lisäksi mahdollinen turvallisuuden suunnittelun ja ketterän kehittämisen prosessin erottaminen.

Rooleilla tarkoitetaan menetelmässä esitettyjä tehtäviä ja niihin kuuluvia vastuita, kuten turvallisuusvastaavaa, kehittäjää tai asiakasta. Roolit voivat prosessimallin tavoin perustua olemassa olevien menetelmien rooleihin, jolloin

tarkastellaan, mitä uusia vastuita rooleille on mahdollisesti lisätty. Uusien roolien osalta tarkastellaan asetettujen vastuiden lisäksi roolin suhteita muihin rooleihin. Keskeistä roolien toiminnassa on myös niiden liittyminen menetelmässä esitettyihin käytänteisiin, tuotoksiin ja kehitysprosessiin.



KUVIO 12 Tutkimuksen viitekehys

Käytänteillä tarkoitetaan sitä, millaisia turvallisuuden suunnittelun käytänteitä esitetään osaksi ketterää kehittämistä tai laajennetaanko olemassa olevia ketterän kehittämisen käytänteitä paremmin turvallisuuden suunnittelua tukevaksi. Esimerkkejä käytänteistä ovat hyökkäyspuut ja väärinkäyttötapaukset. Käytänteiden vastuut ja prosessin vaihe, jossa niitä toteutetaan, ovat tärkeä osa käytänteiden arviointia.

Tuotokset liittyvät yleensä käytänteisiin, jonka vuoksi niitä tarkastellaan yhdessä tässä tutkielmassa. Tuotos voi olla mikä tahansa malli, kuvaus tai ohjelmiston osa, joka tuotetaan osana järjestelmän kehittämistä. Turvallisuuden suunnittelu vaatii tuekseen enemmän tuotoksia kuin ketterä kehittäminen, joten tuotosten toteutustapaan ja niiden määrään kiinnitetään tarkastelussa erityistä huomiota.

Viitekehyksellä on tarkoitus jäsentää ketterän kehittämisen ja turvallisuuden suunnittelun integroinnin osa-alueiden lisäksi niiden painottumista joko turvallisuuteen tai ketteryyteen (ks. kuvio 12). Ihannetapauksessa prosessi, roolit, käytänteet ja tuotokset kohtaavat molempien osalta, jolloin voidaan saavuttaa järjestelmän riittävä turvallisuustaso rikkomatta ketteriä periaatteita. Käytännössä integrointi on kuitenkin niin turvallisuuden kuin ketteryyden osalta tietylle tasolle asetettava kompromissi, jonka painopiste vaihtelee eri menetelmissä. Integrointiesityksen yksittäisten osa-alueiden painotukset määrittävät lopulta sen, onko menetelmä ketterä ja voidaanko sillä tuottaa samalla turvallisia järjestelmiä.

5.2 Viitekehykseen perustuva vertailu ja analysointi

Seuraavissa kohdissa vertaillaan ja analysoidaan tutkimuksen viitekehykseen perustuen turvallisuuden suunnittelun ja ketterän kehittämisen integrointiesityksiä. Ensiksi esityksiä tarkastellaan prosessin kannalta. Toiseksi pohditaan prosessiin osallistuvia rooleja, ja kolmanneksi käydään läpi käytänteitä sekä tuotoksia.

5.2.1 Prosessi

Menetelmien kehittämisprosesseista on löydettävissä yhteisiä piirteitä, ja niitä voidaan luokitella eri kategorioihin. Yleisin näistä on *aikaisempaan menetelmään perustuva prosessi*, jolloin integrointiesityksen perustana on käytetty jotain ketterän kehittämisen menetelmää. Esimerkkejä tällaisista esityksistä ovat alaluvussa 4.4 esitellyt XP :n räätälöinnit (Boström ym., 2006 ; Beznosov, 2003) ja Scrumiin perustuvat esitykset (Azham ym., 2011; Mougouei ym., 2013). Muita esimerkkejä aikaisempaa prosessia hyödyntävistä integrointiesityksistä ovat Aydalin, Paigen, Chiversin ja Brooken (2006), Musan, Norwawin, Selamatin ja Sharifin (2011), Gen, Paigen, Polackin, Chiversin ja Brooken (2006) sekä Gen ym. (2007) esitykset. Aikaisempaan menetelmään perustuvassa prosessimallissa lähestymistapana on yleensä tietty prosessin osa-alue, jota laajennetaan turvallisuuden suunnittelun käytänteillä. Perustana olevan menetelmän prosessi pysyy näin suurelta osin alkuperäisenä, jolloin laajennettu menetelmä on helppo ottaa käyttöön ja integroida osaksi ketterää kehittämistä. Toisaalta integroinnin tarkastelu vain tietystä kehittämisprosessin osa-alueesta voi olla liian suppea, eikä kaikille prosessin vaiheille osoiteta turvallisuuden suunnittelun vaatimia käytänteitä tai prosessissa vaadittavia muutoksia. Esimerkiksi Boström ym. (2006) keskittyvät XP:n suunnittelupelin laajentamiseen, ja painopiste on näin ollen kehittämisprosessin vaatimusmäärittelyssä. Kuten Boström ym. (2006) mainitsevat, suunnittelupelin laajentaminen vaikuttaa kuitenkin myös XP:n prosessin muihin aktiviteetteihin. He mainitsevat esimerkkinä iteraation toteuttamisvaiheen, jossa kehittäjät kirjoittavat testejä ja toteuttavat valitut käyttäjäkertomukset. Turvallisuustoimintojen osalta esimerkiksi testien luonne voi olla hyvin erilainen, jolloin täydentävät menetelmät, kuten haavoittuvuustestit voisivat olla tarpeellisia (Boström ym., 2006).

Scrumiin pohjautuvista prosessimalleista Azham ym. (2011) ovat keskittyneet Boströmin ym. (2006) tavoin tietyn käytänteen laajentamiseen osana tavallista kehittämisprosessia. Azhamin ym. (2011) esittämä laajennus turvallisuuden kehitysjonosta tuo turvallisuuden suunnittelun yksinkertaisella tavalla osaksi Scrumin ketterää kehittämisen prosessia. Muista vastaavista esityksistä poiketen Azhamin ym. (2011) turvallisuuden suunnittelun prosessia voidaan pitää osittain erillisenä, sillä turvallisuuden kehitysjonoa hallitaan tavallisista kehitysjonoista erillään. Sen linkittyminen tavalliseen kehitysjonoon ja Scrum-prosessiin tulee kuitenkin selkeästi esille.

Alkuperäiseen prosessiin ei välttämättä esitetä lainkaan muutoksia, jolloin integrointiesityksessä otetaan yleensä kantaa vain prosessin eri vaiheissa toteutettaviin turvallisuuden suunnittelun käytänteisiin ja rooleihin. Tällaisesta lähestymistavasta esimerkkinä on Beznosovin (2003) XP:hen perustuva integrointiesitys. Vaikka lähtökohtana on Beznosovin (2003) mukaan ”riittävä turvallisuustaso” määrittelemättä etukäteen mikä se on, herättää tämä kuitenkin kysymyksen siitä, onko tavallisen ketterän kehittämisen prosessi riittävä kattamaan turvallisuuden suunnittelun vaatimukset. Esimerkiksi suunnittelupelissä Beznosov (2003) käsittelee turvallisuusvaatimuksia käytännössä tavallisina vaatimuksina ja suunnittelupelin prosessiin ei esitetä muutoksia. Tämä ei Boströmin ym. (2006) riitä kattamaan turvallisuusvaatimuksia.

Mougouei ym. (2013) ottavat laajemmin kantaa Scrum-prosessin räätälöintiin, ja he käsittelevät koko kehitysprosessia. Turvallisuuden suunnittelua ei toteuteta erillisenä prosessina, vaan se on integroitu muokattuun Scrum-prosessiin. Prosessia laajennetaan turvallisuuden suunnittelun kahdella ”piikillä”, joissa toteutetaan turvallisuusanalyysi ja -suunnittelu. Vaikka Mougouei ym. (2013) esittävät kokonaisen turvallisuuden suunnittelua tukevan Scrum-prosessin, jää prosessiin liittyvien roolien, käytänteiden ja tuotoksien tarkastelu kuitenkin varsin pintapuoliseksi. Lähestymistavan vahvuutena on kuitenkin kehittämisprosessin parempi tuki turvallisuuden suunnittelulle.

Aikaisempaan menetelmään perustuvan prosessimallin lisäksi integrointiesityksistä on löydettävissä myös muita lähestymistapoja. Siponen ym. (2005) sekä Baca ja Carlsson (2011) esittävät *yleistä prosessia*. Se ei esitä tiettyä prosessimallia, mutta sisältää kuitenkin kehittämisen eri osa-alueet, jotka ovat vaatimusmäärittely, suunnittelu, toteutus ja testaus. Jokaiseen osa-alueeseen esitetään ketterän kehittämisen kanssa yhteensopivia turvallisuuden suunnittelun käytänteitä. Kuten Siponen ym. (2005) mainitsevat, vaiheet eivät ole välttämättä peräkkäisiä ja jokainen vaihe on valinnainen. Näin esitettyjä käytänteitä voidaan joustavasti integroida eri ketterän kehittämisen menetelmiin tai esimerkiksi organisaatiokohtaiseen prosessimalliin. Siponen ym. (2005) käyttävät prosessin soveltamisessa esimerkkinä FDD (Feature Driven Development) – menetelmää (Palmer & Felsing, 2001) ja Baca ja Carlsson (2011) organisaatiokohtaista ketterän kehittämisen prosessia. Monet kehittämisen eri osa-alueisiin esitetyistä käytänteistä ovat yhteneväisiä, mutta malleissa on myös eroavaisuuksia, joita käsitellään käytänteitä käsittelevässä kohdassa 5.2.3. Prosessin kannalta tarkasteltuna turvallisuuden suunnittelun vaatimukset eri kehittämisen vaiheille tulevat hyvin esille molemmissa menetelmissä.

Siposen ym. (2005) sekä Bacan ja Carlssonin (2011) lisäksi myös Bansalin ja Jollyn (2014) esitys voidaan luokitella yleisen prosessin kategoriaan, joskin he keskittyvät lähinnä turvallisuuden suunnittelun aktiviteettien yhteensopivuuden matemaattiseen tarkasteluun eri kehitysvaiheissa. Koska esityksen lähestymistapa ketterän kehittämisen ja turvallisuuden suunnittelun integrointiin on hieman erilainen, ei se ole täysin vertailukelpoinen Siposen ym. (2005) sekä Bacan ja Carlssonin (2011) esityksiin.

Prosessimalli voi myös koostua ketterän kehittämisen ja turvallisuuden suunnittelun menetelmien yhdistelmästä. *Integroiva prosessi* sisältää ketterän kehittämisen periaatteiden mukaiset ohjelmistokehityksen käytänteet, jossa turvallisuuden suunnittelu on yhdistetty kehittämisprosessiin. Esimerkkinä tällaisesta prosessimallista ovat Singhalin ja Singhalin (2011) sekä Othmanen, Angin, Weffersin ja Bhargavan (2013) integrointiesitykset. Lähestymistavassa on monia hyviä puolia. Se huomioi turvallisuuden suunnittelun käytänteiden, kuten turvallisuusvaatimusten, uhkien mallintamisen, toteutuksen ja testauksen vaatimat muutokset kehitysprosessiin. Eri menetelmien hyväksi havaittuja käytänteitä voidaan yhdistää samaan prosessiin, jolloin sekä turvallisuuden suunnittelu että ketterä kehittäminen voidaan huomioida mahdollisimman kattavasti. Toisaalta prosessi voi muodostua ketterän kehittämisen kannalta liian laajaksi, jos se sisältää paljon turvallisuuden suunnittelun käytänteitä tai tuotoksia. Esimerkiksi Singhalin ja Singhalin (2011) viitekehyksessä tuotetaan ensimmäisessä iteraatiossa kattava turvallisuuden viitekehys ja toteutetaan laajasti turvallisuusuhkien mallintamista ja testausta, mikä vähentää kehitysprosessin ketteryyttä. Myös Othmanen ym. (2013) prosessia voidaan pitää melko raskaana, ja kuten he itsekin mainitsevat, sen rajoituksia ovat skaalautuvuuden puute ja ylimääräiset kustannukset.

Monissa ketterän kehittämisen ja turvallisuuden suunnittelun integrointiesityksissä ei esitetä käytettäväksi tiettyä kehittämisprosessia. Nämä esitykset sisältävät yleensä tietyn ketterään kehittämiseen soveltuvan turvallisuuden suunnittelun käytänteen tai joukon käytänteitä sekä käytänteisiin liittyvät roolit. Käytänteet ja roolit voivat kattaa koko kehitysprosessin tai vain osan siitä. Esimerkiksi Peetersin (2005) väärinkäytön kertomukset keskittyvät prosessin kannalta lähinnä vaatimusmäärittelyyn, eivätkä tarjoa yksittäisenä käytänteenä ratkaisua turvallisuuden suunnittelun ja ketterän kehittämisen integrointiin. Kongslin (2006) esittämät käytänteet puolestaan liittyvät kehitysprosessin eri vaiheisiin vaatimusmäärittelystä testaukseen, mutta käytettävää prosessia ei esitetä. Käytänteitä voidaan hyödyntää tehokkaina apuvälineinä osana integrointia, mutta ne vaativat kuitenkin tuekseen jonkin turvallisuuden suunnittelua tukeva ketterän kehittämisen prosessin tai viitekehyksen. Muita samaan kategoriaan luokiteltavia, integrointia käsitteleviä teoksia ovat julkaisseet muun muassa Tappenden, Beatty, Miller, Geras ja Smith (2005) sekä Keramati ja Mirian-Hosseinabadi (2008).

Taulukossa 5 on esitetty tiivistelmä kirjallisuuskatsauksen tuloksena löydetyistä menetelmistä ja käytänteistä prosessin kannalta. Vaikka integrointiesitykset on luokiteltu prosessin näkökulmasta eri kategorioihin, ovat rajat kuitenkin häilyviä. Esimerkiksi yleiseen kategoriaan kuuluvia esityksiä voidaan soveltaa osana aikaisempaa menetelmää, kuten XP:tä tai Scrumia. Vastaavasti monissa esityksissä ei oteta kantaa kehitysprosessiin, jolloin esitettyjä käytänteitä voidaan soveltaa periaatteessa mihin tahansa prosessimalliin. Luokittelun tarkoituksena on kuitenkin muodostaa yleiskuva integrointiesitysten prosessimallien pääasiallisesta lähestymistavasta, eikä käsitellä yksityiskohtaisesti kaikkia mahdollisia käytettävien prosessimallien variaatioita.

TAULUKKO 5 Menetelmät ja käytänteet prosessin kannalta

Prosessin tyyppi	Kuvaus	Lähteet
Aikaisempaan menetelmään perustuva prosessi	Räätälöi perustana olevan ketterän menetelmän prosessia tai esittää turvallisuuden suunnittelun käytänteitä olemassa olevaan prosessiin	Boström ym., 2006; Beznosov, 2003; Azham ym., 2011; Mougouei ym., 2013; Aydal ym., 2006; Musa ym., 2011; Ge ym., 2006; Ge ym., 2007
Yleinen prosessi	Kattaa kehittämisen eri osa-alueet yleisellä prosessilla tai tarkastelee osa-alueiden turvallisuuden suunnittelun käytänteitä	Siponen ym., 2005; Baca & Carlsson, 2011; Bansal & Jolly, 2014
Integroiva prosessi	Esittää turvallista ketterän kehittämisen prosessi, joka ei perustu tiettyyn aiempaan prosessimalliin	Singhal & Singhal, 2011; Othmane ym., 2013
Ei esitä prosessia	Esittää käytänteen tai joukon ketterään kehittämiseen soveltuvia käytänteitä	Peeters, 2005; Kongsli, 2006; Keramati & Mirian-Hosseinabadi, 2008; Tappenden ym., 2005

5.2.2 Roolit

Ihmisten rooli kehitystyössä on ratkaiseva (Jacobson ym., 1999) ja sama pätee myös turvallisuuden suunnittelun ja ketterän kehittämisen integrointiin. Integroinnissa on tärkeää huomioida paitsi liiketoiminnan kannalta merkittävät suojattavat kohteet, myös turvallisuuden suunnittelun ja kehitystyön tekninen luonne. Ensin mainitun osalta lienee perusteltua väittää, että asiakkaalla on paras näkemys suojattavien kohteiden tunnistamisessa. Kehittäjät tai turvallisuusasiantuntijat puolestaan tuntevat asiakasta paremmin mahdolliset turvallisuusuhkat teknisestä näkökulmasta. Kolme keskeistä roolia, turvallisuusasiantuntija, kehittäjä sekä asiakas esiintyvät eri painoarvoilla turvallisuuden suunnittelun ja ketterän kehittämisen integrointiesityksissä.

Turvallisuusasiantuntijoita tai turvallisuusvastaavaa hyödynnetään muun muassa Singhalin ja Singhalin (2011), Boströmin ym. (2006), Beznosovin (2003) sekä Azhamin ym. (2011) esityksissä. Boström ym. (2006) laajentavat ketterän kehittämisen rooleja yhdellä tai useammalla turvallisuuden suunnittelijalla, joka tukee turvallisuuden kehittämisessä niin asiakasta kuin kehittäjiäkin. Turvallisuuden suunnittelijan pääasiallisena tehtävänä on tukea asiakasta turvallisuuden vaatimusmäärittelyssä, ja kehittäjiä erityisesti toteutuksen aikana turvallisuusvalmennuksella ja pariohjelmoinnissa (Boström ym., 2006). Azham ym. (2011) laajentavat Scrumin rooleja turvallisuusvastaavalla, jonka tehtäviä ovat muun muassa turvallisuustoimintojen tunnistaminen, riskien dokumentointi ja

turvallisuustestaus. Singhal ja Singhal (2011) hyödyntävät turvallisuuden asiantuntijoita väärinkäytön kertomusten laadinnassa ja uhkien mallintamisessa, mutta he eivät esitä kehitystiimiin varsinaista turvallisuusvastaavan roolia. Beznosov (2003) puolestaan käyttää turvallisuuden suunnittelijoita kehittämisen eri vaiheissa, kuten käyttäjäkertomusten laadinnassa, pariohjelmoinnissa ja testauksessa. Edellä mainittujen lisäksi Kongsli (2006) mainitsee, että tiimin turvallisuustietoisuuden lisäämisestä huolimatta turvallisuusasiantuntijan tarvetta ei voida kokonaan sulkea pois. Myös Gen ym. (2007) mukaan tiimin tietoisuus ei tavallisesti riitä korvaamaan turvallisuusasiantuntijan tarvetta, joka toimii syvällisellä turvallisuustietämyksellään kehitystiimin turvallisuusresurssina.

Turvallisuusasiantuntijan tai –asiantuntijoiden roolin painoarvo siis vaihtelee niiden esitysten välillä, joissa rooleja laajennetaan turvallisuuden suunnittelun asiantuntemuksella. Yhteistä näille on kuitenkin se, että ketterän kehittämisen roolien lisäksi myös turvallisuuden asiantuntemus on nähty tarpeelliseksi. Kehittäjillä tai asiakkaalla ei välttämättä ole syvällistä turvallisuustietämystä, ja turvallisuusasiantuntijan rooli on tässä suhteessa perusteltua. He voivat tuoda kehittämisen eri vaiheisiin erityisesti turvallisuuden teknistä tietämystä, joka auttaa turvallisuustoimintojen kehittämisessä. Kongsli (2006) nostaa kuitenkin esille tärkeän näkökulman, jonka mukaan turvallisuusasiantuntijan ei tulisi yksin omistaa turvallisuuden suunnittelua vaan toimia tiimiä ohjaavana valmentajana. Kongsli (2006) laajentaa XP:n yhteisomistajuutta näin myös turvallisuuden suunnitteluun; turvallisuus ei ole tietyn roolin vastuulla, vaan siihen voivat vaikuttaa osaltaan kaikki kehitystyössä mukana olevat henkilöt. Kuten Singhalin ja Singhalin (2011), Boströmin ym. (2006) sekä Beznosovin (2003) esimerkit osoittavat, turvallisuusasiantuntija on tuotu osaksi kehittämistä juuri tukevana roolina, jolloin hän ei vastaa yksin turvallisuudesta. Azham ym. (2011) esittävät kuitenkin muista poiketen turvallisuusvastaavalle hieman aktiivisempaa roolia, sillä hän hallitsee yksin turvallisuuden kehitysjonoa, ja toteuttaa turvallisuustestauksen. Vaikka tässä tarvitaankin vuorovaikutusta asiakkaaseen ja kehittäjiin, painottavat Azham ym. (2011) melko yksipuolisesti turvallisuusvastaavan roolia.

Kehittäjät osallistuvat ketterälle kehittämiselle tyypillisten tehtävien lisäksi turvallisuuden suunnittelun eri vaiheisiin käytännössä kaikissa integrointiesityksissä. Tämä on luonnollista, sillä kehittäjät lopulta toteuttavat turvallisuustoiminnot. Kehittäjät osallistuvat turvallisuustoimintojen toteutuksen ja niiden suunnittelun lisäksi muun muassa väärinkäytön kertomusten tai turvallisuuteen liittyvien käyttäjäkertomusten laadintaan (Peeters, 2005; Singhal & Singhal, 2011; Boström ym., 2006; Kongsli, 2006; Beznosov, 2003), riskien tunnistamiseen ja uhkien mallintamiseen (Singhal & Singhal, 2011; Siponen ym., 2005) sekä testaukseen (Singhal & Singhal, 2011; Kongsli, 2006; Siponen ym., 2005; Baca & Carlsson, 2011). Yleisenä johtopäätöksenä voidaan todeta, että kehittäjillä on turvallisuuden suunnittelussa aktiivinen rooli esitetyistä käytänteistä riippumatta. Turvallisuusasiantuntijan käyttö ei muuta merkittävästi kehittäjien tehtäviä, sillä hänen roolinsa on tyypillisesti tukea kehittämistä turvalliseen suuntaan yhteistyössä kehittäjien kanssa.

Myös *asiakas* osallistuu turvallisuuden suunnitteluun erityisesti turvallisuustoimintojen vaatimusmäärittelyssä. Suojattavien kohteiden tunnistaminen on yleinen turvallisuuden suunnittelun aktiviteetti, jossa asiakas on keskeisessä roolissa. Tässä käytetään yleisesti apuna käyttäjäkertomuksia, väärinkäytön kertomuksia tai käyttötapausten eri muotoja. Suojattavien kohteiden tunnistaminen on mainittu muun muassa Siposen ym. (2005), Singhalin ja Singhalin (2011), Boströmin ym. (2006), ja Peetersin (2005) integrointiesityksissä. On huomioitavaa, että kaikissa näistä aktiviteetti toteutetaan asiakkaan ja kehittäjien yhteistyönä, mikä tukee myös ketterän kehittämisen periaatteita. Asiakkaan liiketoiminnan tuntemuksen hyödyntäminen turvallisuuden suunnittelussa on vastaavasti huomioitu myös Musan ym. (2011) ja Othmanen ym. (2013) integrointiesityksissä.

Suojattavien kohteiden tunnistaminen ja käyttäjäkertomusten laatiminen ovat keskeisimmät turvallisuuden suunnitteluun liittyvät asiakkaan tehtävät, mutta turvallisuuden lisääminen ketterään kehittämiseen ei kuitenkaan poista tavallisia asiakkaan rooleja. Kuten Beznosov (2003) mainitsee, asiakas on aktiivisesti kehittämisprosessissa mukana ja muun muassa neuvottelee julkaisuun valittavista käyttäjäkertomuksista, selventää tai parantelee niitä, tuottaa kehitystyössä tarvittavat lisätiedot ja kehittää toiminnalliset testit tai auttaa niissä. Vaikka näitä asiakkaan tehtäviä ei erityisesti mainita integrointiesityksissä, ovat ne kuitenkin mukana ketterän kehittämisen käytänteiden kautta.

5.2.3 Käytänteet ja tuotokset

Turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämisen keskeisen osan muodostavat käytänteet ja niiden tuotokset. Esitetyt käytänteet vaihtelevat sellaisenaan sovellettavista, yksinkertaisista ketterän kehittämisen käytänteistä turvallisuuden suunnittelun yksityiskohtaisempiin käytänteisiin. Tyypillisesti ketterän kehittämisen käytänteitä on myös räätälöity turvallisuuden suunnitteluun paremmin sopivaksi. Seuraavaksi tarkastellaan yleisimpiä esitettyjä käytänteitä sekä vertaillaan niiden eroja ja yhteneväisyyksiä.

Käyttäjäkertomukset ovat ketterän kehittämisen yleinen käytänne, joilla mallinnetaan järjestelmän toiminnallisuutta. Yksittäisistä ketterän kehittämisen käytänteistä niitä on sovellettu eniten myös turvallisuuden suunnitteluun. Muun muassa Peetersin (2005) esittämiä *väärinkäytön kertomuksia* käytetään esimerkiksi Singhalin ja Singhalin (2011), Kongslin (2006) sekä Boströmin ym. (2006) integrointiesityksissä. Väärinkäytön kertomuksista on käytetty eri termejä, ja kertomuksia on käsitelty eri tarkkuustasoilla. Periaatteena on kuitenkin esityksestä riippumatta kuvata sitä, kuinka mahdollinen hyökkääjä voi käyttää järjestelmää väärin ja tätä kautta vaarantaa suojattavia kohteita. Kehittämisprosessissa se liittyy erityisesti vaatimusmäärittelyyn, mutta vaikuttaa tätä kautta kaikkiin kehittämisprosessin vaiheisiin. Tyypillisesti käyttäjät ja kehittäjät laativat yhteistyössä väärinkäytön kertomukset, ja niiden laatimiseen voi osallistua myös turvallisuusasiantuntija Boströmin ym. (2006) esityksen mukaisesti. Käytänne tuottaa luonnollisella kielellä kuvauksen hyökkääjän toiminnasta.

Väärinkäytön kertomukset ovat yksinkertainen, ketterään kehittämisen periaatteiden mukainen käytänne, jolla mahdollisia turvallisuusuhkia voidaan kuitenkin mallintaa useisiin tilanteisiin riittävällä tasolla. Kertomusten pisteyttäminen liiketoiminnan haitallisuuden perusteella Peetersin (2005) esityksen mukaisesti toimii samalla järjestelmän riskien arvioinnin käytänteenä ja haavoittuvuuden arvioinnin työkaluna. Osaltaan se rakentaa luottamusta järjestelmään tuottamalla kertomusten kautta dokumentaatiota kehitetyistä turvallisuusmekanismeista. Nämä ovat ominaisuuksia, jotka ottavat huomioon useita alaluvussa 2.2 esitettyjä SSE-CMM (ISO/IEC 21827, 2008) –standardin turvallisuuden suunnittelun peruskäytänteitä. On selvää, että lyhyillä ja epäformaaleilla kuvauksilla turvallisuutta ei voida mallintaa kattavasti, mutta yksittäisenä ketterään kehittämiseen sopivana käytänteenä väärinkäyttötapaukset toimivat tehokkaana työkaluna.

Väärinkäytön kertomusten lisäksi käyttäjäkertomuksia on sovellettu *turvallisuuden käyttäjäkertomuksissa*. Suppeammin jäsenneltynä turvallisuuden käyttäjäkertomuksilla voidaan tarkoittaa Beznosovin (2003) esityksen mukaista käyttäjäkertomusta, jolla mallinnetaan toteutettava turvallisuuden toiminnallisuus. Tämä vastaa käytännössä normaalin käyttäjäkertomuksen soveltamista sellaisenaan turvallisuustoimintoihin. Boströmin ym. (2006) mukaan turvallisuuden käyttäjäkertomukset ovat toiminnallisia turvallisuusvaatimuksia väärinkäytön kertomuksissa osoitettujen uhkien vastatoimien toteuttamiselle. Turvallisuuden käyttäjäkertomuksille voidaan suorittaa testausta, toisin kuin väärinkäytön kertomuksille (Boström ym., 2006). Vastaavasti turvallisuuden käyttäjäkertomuksia ovat jäsennelleet myös Othmane ym. (2013). Vaikka turvallisuuden käyttäjäkertomuksissa on lähteestä riippuen käsitteellisiä eroja, oleellista käytänteessä on kuitenkin *vastatoimen* huomioiminen käyttäjäkertomusten avulla. Näin voidaan tunnistaa mahdollisten uhkien lisäksi myös niihin liittyvät turvallisuuden toiminnalliset vaatimukset. Vastatoimi voi olla huomioituna myös väärinkäytön kertomuksissa (esim. Singhal & Singhal, 2011), joten turvallisuuden käyttäjäkertomuksilla ja väärinkäytön kertomuksilla voidaan tarkoittaa osittain samaa käytännettä.

Väärinkäyttötapaukset esiintyvät yhtenä turvallisuuden suunnittelun käytänteenä muun muassa Othmanen ym. (2013), Bacan ja Carlssonin (2011), Musan ym. (2011) sekä Siposen ym. (2005) esityksissä. Alaluvussa 2.4 kuvatuilla väärinkäyttötapauksilla voidaan mallintaa tavallisten käyttötapausten tapaan järjestelmän ja käyttäjien vuorovaikutus, mutta vuorovaikutuksen lopputulos on haitallinen. Väärinkäytön kertomusten tavoin väärinkäyttötapaukset liittyvät kehittämisprosessissa erityisesti järjestelmän vaatimusmäärittelyyn, ja niiden laatimiseen osallistuvat tyypillisesti kehittäjät ja asiakas. Väärinkäyttötapaukset tuottavat yleisellä käyttötapausten notaatiolla laaditun diagrammin ja väärinkäyttötapausten kuvaukset.

Väärinkäyttötapauksia käsitellään integrointiesityksissä eri tarkkuustasoilla. Esimerkiksi Othmane ym. (2013) mainitsevat sen mahdollisena käytänteenä alkuvaiheen uhkien mallinnuksessa, mutta väärinkäyttötapaukset eivät ole keskeisessä roolissa kehittämisprosessia. Baca ja Carlsson (2011) esittävät vas-

taavasti väärinkäyttötapauksia käytettäväksi kehittämisprosessin eri vaiheissa ilman käytänteen yksityiskohtaisempaa tarkastelua. Musa ym. (2011) ja Siponen ym. (2005) sen sijaan havainnollistavat väärinkäyttötapausten soveltamista yksityiskohtaisemmin. Molemmissa väärinkäyttötapauksia hyödynnetään myös suunnitteluvaiheessa, minkä myös Baca ja Carlsson (2011) näkevät tarpeelliseksi. Siponen ym. (2005) muokkaavat väärinkäyttötapausten kuvauksia muun muassa vastatoimilla ja väärinkäyttötapauksiin liittyvän iteraation tiedoilla. Nämä ovat tarpeellisia laajennuksia, sillä tarvittava vastatoimi voidaan liittää tällä tavoin osaksi uhkaa, ja toisaalta on tärkeää tietää, missä vaiheessa kehittämistä väärinkäyttötapaus toteutetaan.

Boströmin ym. (2006) mukaan väärinkäyttötapaukset muistuttavat väärinkäytön kertomuksia, mutta ne ovat muodollisempia, ja tätä kautta niitä on vaikeampi soveltaa osana XP:n käytänteitä. On totta, että väärinkäytön kertomukset ovat helpommin integroitavissa ketterään kehittämiseen, mutta toisaalta turvallisuuden suunnittelu vaatii myös muodollisuutta. Väärinkäyttötapaukset tukevat tässä suhteessa paremmin turvallisuuden suunnittelua. Käytänteenä väärinkäyttötapaukset eivät ole kuitenkaan ristiriidassa ketterän kehittämisen kanssa, ja niillä voidaan osoittaa väärinkäytön kertomusten tavoin monia turvallisuuden suunnittelun aktiviteetteja. Niiden avulla voidaan mallintaa uhkia sekä määritellä niiden tarvittavat vastatoimet, ja tuotetut kuvaukset palvelevat osaltaan myös turvallisuuden varmistamista.

Alaluvussa 2.5 tarkemmin kuvattuja *hyökkäyspuita* hyödynnetään esimerkiksi Singhalin ja Singhalin (2011) viitekehyksessä ja Kongslin (2006) turvallisuuden suunnittelun käytänteissä. Ne mallintavat hyökkäysten vaikutuksia järjestelmään ja tuottavat diagrammin tai hierarkkisesti muodostetun tekstimuotoisen kuvauksen mahdollisesta hyökkäyksestä. Singhal ja Singhal (2011) ja Kongsli (2006) käyttävät hyökkäyspuita samalla periaatteella täydentämään väärinkäytön kertomuksia erityisesti monimutkaisemmissa tapauksissa. Singhalin ja Singhalin (2011) mukaan hyökkääjät pyrkivät hyödyntämään muita kuin käyttäjän sisäänpääsykohtia järjestelmässä, jolloin hyökkäystapaa ei kaikissa tilanteissa pystytä mallintamaan väärinkäytön kertomuksilla. Toisaalta hyökkäyspuut eivät määrittele uhkan välttämisen strategiaa tai uhkan hetkellä vallitsevaa tilannetta, jotka voidaan osoittaa väärinkäytön kertomuksilla (Singhal & Singhal, 2011).

Hyökkäyspuut täydentävät siis edellä kuvattuja väärinkäytön kertomusten tai turvallisuuden käyttäjäkertomusten puutteita, ja niillä pystytään mallintamaan hyökkäyksiä käyttäjäkertomuksia yksityiskohtaisemmin. Niiden avulla voidaan löytää käyttäjäkertomuksesta useita haavoittuvuuksia (Kongsli, 2006), mikä puoltaa turvallisuuden suunnitteluun esitettyjen käyttäjäkertomusten täydentämistä hyökkäyspuilla. Ketteryyden näkökulmasta tarkasteltuna käytänne on yksinkertainen, mutta toisaalta hyökkäyspuilla mallinnettavat käyttäjäkertomukset ja mallintamisen tarkkuus tulisi harkita tarkkaan. Hyökkäyspuilla voidaan laajimmillaan kattaa monia hyökkäyksen variaatioita, mutta mallintaminen vie tällöin aikaa eikä tue ketteryyden periaatteita.

Roolimatriisilla voidaan tunnistaa järjestelmän käyttäjäroolit ja niiden pääsyoikeudet järjestelmään (Howard & Lipner, 2009), ja käytänne esiintyy muun muassa Bacan ja Carlssonin (2011) sekä Bansalin ja Jollyn (2014) esityksissä. Roolimatriisi auttaa Bacan ja Carlssonin (2011) mukaan käyttäjäkertomusten kirjoittamisessa. He kuitenkin huomauttavat, että uusien tuotteiden kehittämisessä käyttäjäroolien tunnistaminen voi olla liian aikaista vaatimusmäärittelyssä. Bansal ja Jolly (2014) ovat poimineet roolimatriisin yhdeksi tärkeäksi turvallisuuden suunnittelun aktiviteetiksi, mutta he eivät perustele tarkemmin valintaa. Vaikka käytänne ei ole yleisimpien esitettyjen käytänteiden joukossa ja sitä on käsitelty melko suppeasti, voi roolimatriisi olla hyödyllinen esimerkiksi laajempien järjestelmien kehittämisessä. Jos kehitettävässä järjestelmässä on vähän käyttäjiä ja pääsyoikeustasoja, ei roolimatriisin käytölle ole juuri perusteita.

Koodaussäännöt ovat ketterän kehittämisen käytänne, joka esiintyy monissa turvallisuuden suunnitteluun integrointiesityksissä (esim. Baca & Carlsson, 2011; Boström ym., 2006; Bansal & Jolly, 2014; Musa ym. 2011). Muiden käytänteiden tavoin koodaussäännöt esiintyvät lähteestä riippuen eri nimillä, eikä käytänteen soveltamista ole yleensä käsitelty tarkemmin. Boströmin ym. (2006) mukaan osaa turvallisuusuhkista täytyy käsitellä koko järjestelmän kattavilla tekniikoilla, johon koodaussäännöt soveltuvat hyvin. Käytänne liittyy siis kehittämisprosessin osa-alueista erityisesti toteutukseen, ja koodaussäännöt ovat tähän hyvä kehittäjien apuväline turvallisuuden huomioimiseen. Musa ym. (2011) nostavat esille turvallisuusarkkitehtuurin parantamisen lisäksi koodaussääntöjen eduiksi yhteisomistajuuden tukemisen.

Koodaussääntöjen lisäksi keskeisesti toteutukseen liittyvä käytänne, *staattisen koodin analyysi* esiintyy mahdollisena käytänteenä muun muassa Bacan ja Carlssonin (2011); Boströmin ym. (2006) sekä Bansalin ja Jollyn (2014) esityksissä. Sillä tarkoitetaan koodin tarkastelua virheiden tunnistamiseksi esimerkiksi soveltuvilla työkaluilla (McGraw, 2006). Staattisen koodin analyysin soveltamista ei käsitellä koodaussääntöjen tavoin integrointiesityksissä tarkemmin, mutta Boström ym. (2006) mainitsevat käytänteen soveltuvan aikaisemman koodin analysointiin uusien koodaussääntöjen käyttöönotossa.

Ketterän kehittämisen ja erityisesti XP:n käytänteistä turvallisuuden suunnitteluun yhteensopiviksi mainitaan koodaussääntöjen lisäksi muun muassa pariohjelmointi ja jatkuva integrointi (Beznosov, 2003; Boström ym., 2006; Musa ym., 2011) sekä pienet julkaisut (Beznosov, 2003; Musa ym., 2011). *Pariohjelmoinnin* käytänne vastaa normaalia pariohjelmointia, jossa Beckin (1999a) mukaan kaksi henkilöä kirjoittaa koodia samalla työasemalla. Turvallisuuden näkökulmasta tavoitteena on varmistaa Musan ym. (2011) mukaan koodin yhteensopivuus muihin turvallisuuden käytänteisiin. Boströmin ym. (2006) esityksessä pariohjelmointiin osallistuu muista poiketen myös turvallisuusvastaava. *Jatkuvassa integroinnissa* uutta koodia lisätään ja integroidaan järjestelmään useita kertoja päivässä (Beck 1999a), mikä auttaa Beznosovin (2003) mukaan täydentämään turvallisuuden suunnittelijoiden käsitystä asiakkaan ympäristöstä, sekä lisää turvallisuutta ennen sen varsinaista tuotantokäyttöä. *Pienillä julkaisuilla* turvallisuuden suunnittelijat voivat puolestaan käyttää edellisten iteraati-

oiden kokemuksia uusien turvallisuusmekanismien kehittämisessä (Beznosov, 2003). Pariohjelmointi, jatkuva integrointi sekä pienet julkaisut ovat hyviä esimerkkejä turvallisuuden suunnittelua tukevista yksinkertaisista ketterän kehittämisen käytänteistä.

Kuten ketterässä kehittämisessä ja järjestelmäkehityksessä yleensä, on toteutetut toiminnallisuudet testattava, jotta voidaan varmistua ohjelmiston toimivuudesta eri tilanteissa. Turvallisuuden lisääminen ei poista testaamisen tarvetta, ja *turvallisuustestaus* esiintyykin käytänteenä useissa integrointiesityksissä (esim. Tappenden ym., 2005; Singhal & Singhal, 2011; Azham ym., 2011; Kongsli, 2006; Siponen ym., 2005; Beznosov, 2003; Musa ym., 2011). Testausta on käsitelty lähteestä riippuen eri painoarvoilla, mutta valtaosassa turvallisuustestaus on vähintäänkin mainittu tarpeellisena käytänteenä tai se on huomioitu kehitysprosessissa. Testauskäytänteinä esiintyvät yleisesti *hyväksyntätestaus* (mm. Musa ym., 2011; Singhal & Singhal, 2011; Kongsli, 2006), *yksikkötestaus* (mm. Tappenden ym., 2005; Singhal & Singhal, 2011; Kongsli, 2006; Beznosov, 2003) ja *läpäisytestaus* (mm. Singhal & Singhal, 2011; Kongsli, 2006). Muun muassa XP:ssä testaus mainitaan asiakaslähtöiseksi (Beck, 1999a), ja asiakas laatii käyttäjäkertomusten toiminnallisuuksien hyväksyntätestit. Turvallisuuden hyväksyntätestit tehdään yleisesti vastaavalla periaatteella (esim. Beznosov, 2003; Singhal & Singhal, 2011), mutta lähtökohtana ovat väärinkäytön kertomukset. Yksikkötestauksella voidaan Kongslin (2006) mukaan kuvata järjestelmään kohdistuvaa hyökkäystä esimerkiksi tietokantarajapinnan kautta. Läpäisytestauksella puolestaan voidaan löytää haavoittuvuuksia järjestelmästä (Singhal & Singhal, 2011), ja Kongslin (2006) mukaan läpäisytestausta tulisi tehdä varhaisista iteraatioista alkaen kaikissa ympäristöissä. Ketterän kehittämisen tavoin myös turvallisuustestejä esitetään toteutettavaksi soveltuvin osin automaattisesti, jolloin testejä voidaan suorittaa helposti, ja niitä voidaan toistaa useita kertoja (Kongsli, 2006; Singhal & Singhal, 2011).

Testaus on tärkeä osa ketterän kehittämisen ja turvallisuuden suunnittelun integrointia. Yleisesti ottaen testaus on otettu hyvin huomioon integrointiesityksissä, ja se tukee muiden esitettyjen käytänteiden ohella turvallisuuden varmistamista. Jos tarkastellaan esimerkiksi CC:n (CC, 2012) turvallisuuden arvioinnin tasoja (luku 2.3), vaaditaan jo kahdelle alimmalle tasolle pääsemiseksi perusteellista testausta. Integrointiesityksissä yleisesti esiintyvillä hyväksyntä-, yksikkö- ja läpäisytesteillä voidaan kattaa testauksen vaatimukset tasolle 2, jossa vaaditaan järjestelmän rakenteellinen testaus. Osittain myös tason 3 vaatimukset täyttyvät läpäisytestien kautta, sillä tämän tyyppisten hyökkäysten torjuminen tulee CC:n (2012) mukaan jollain tavalla osoittaa.

Turvallisuustietoisuuden lisääminen koko projektitiimissä on tärkeää (Ge ym., 2007), ja *turvallisuuskoulutus* on yksi esitetyistä käytänteistä (Ge ym., 2007; Azham ym., 2011; Bansal & Jolly, 2014). Gen ym. (2007) mukaan vaatimukset vaihtelevat roolien mukaan, sillä sidosryhmien koulutuksen tulee olla yleisemmällä tasolla, kun taas kehittäjien tulee ymmärtää paremmin turvallisuustoimintoja tekniseltä kannalta. Turvallisuuskoulutus on helppo toteuttaa, sillä se ei

vaikuta kehittämisprosessiin tai muihin käytänteisiin, mutta tukee kuitenkin yleistä turvallisuustietoisuutta.

Taulukko 6 esittää yhteenvedon yleisimmistä integrointiesityksissä esiintyvistä käytänteistä. On huomioitava, että taulukossa ei ole mainittu yksittäisiä kirjallisuuskatsauksessa esiintyneitä käytänteitä. Lähteissä on myös käsitteellisiä eroja esimerkiksi väärinkäytön kertomuksissa, mutta käytänteet on yhdistetty niiden piirteiden perusteella samaan kategoriaan. Käytänteitä on tarkasteltu eri tarkkuustasoilla, mutta yhteenvetoon on poimittu kaikki esitykset, joissa käytänne on mainittu tai sovellettu osana kehittämisprosessia.

TAULUKKO 6 Integrointiesitysten käytänteitä

Käytänne	**Kuvaus**	**Lähteet**
Väärinkäytön kertomukset	Käyttäjäkertomusten variaatio, jossa kuvataan, kuinka järjestelmää voidaan käyttää väärin.	Peeters, 2005; Singhal & Singhal, 2011; Kongsli, 2006; Boström ym., 2006
Turvallisuuden käyttäjäkertomukset	Käyttäjäkertomus, joka kuvaa väärinkäytön vastatoimen	Boström ym., 2006; Beznosov, 2003; Othmane ym., 2013
Väärinkäyttötapaukset	Käyttötapausten variaatio, jossa lopputulos on haitallinen järjestelmälle	Othmane ym., 2013; Baca & Carlsson, 2011; Musa ym., 2011; Siponen ym., 2005
Hyökkäyspuut	Mallintavat hyökkäyksen vaikutuksia järjestelmään kuvaamalla mahdollisia hyökkäystapoja	Singhal & Singhal, 2011; Kongsli, 2006
Roolimatriisi	Käyttäjäroolien ja niiden pääsyoikeuksien tunnistaminen	Baca & Carlsson, 2011; Bansal & Jolly, 2014
Koodaussäännöt	Joukko kehittämisessä noudatettavia sääntöjä	Baca & Carlsson, 2011; Boström ym., 2006; Bansal & Jolly, 2014; Musa ym., 2011
Staattisen koodin analyysi	Virheiden tunnistamista kooditasolla esimerkiksi soveltuvilla työkaluilla	Baca & Carlsson, 2011; Boström ym., 2006; Bansal & Jolly, 2014
Pariohjelmointi	Kehittäjien työskentely pareittain	Beznosov, 2003; Boström ym., 2006; Musa ym., 2011;
Jatkuva integrointi	Uuden koodin lisääminen ja integrointi järjestelmään jopa useita kertoja päivässä	Beznosov, 2003; Boström ym., 2006; Musa ym., 2011
Pienet julkaisut	Uusien julkaisujen tuottaminen lyhyellä aikavälillä	Beznosov, 2003; Musa ym., 2011
Turvallisuustestaus	Turvallisuustoimintojen testaus, kuten hyväksyntätestaus, yksikkötestaus tai läpäisytestaus	Tappenden ym., 2005; Singhal & Singhal, 2011; Azham ym., 2011; Kongsli, 2006; Siponen ym., 2005; Beznosov, 2003; Musa ym., 2011
Turvallisuuskoulutus	Turvallisuuden koulutus ja valmentaminen	Ge ym., 2007; Azham ym., 2011; Bansal & Jolly, 2014

5.3 Yhteenveto

Tässä luvussa vertailtiin turvallisuuden suunnittelun ja ketterän kehittämisen integrointiesityksiä. Aluksi esitettiin vertailussa käytetty viitekehys, jonka mukaan integrointiesitysten tarkastelussa kiinnitettiin huomiota prosessiin, rooleihin, käytänteisiin ja tuotoksiin. Seuraavaksi integrointiesityksiä analysoitiin viitekehykseen perustuen.

Kehittämisprosessissa käytetään yleisesti johonkin *aikaisempaan menetelmään perustuvaa prosessia*, kuten XP:n tai Scrumin prosessia. Tällöin jotakin kehittämisprosessin osa-aluetta laajennetaan yleensä turvallisuuden suunnittelun käytänteillä, jolloin perustana olevan ketterän menetelmän prosessi pysyy suurelta osin alkuperäisenä. Muita tapoja hyödyntää aikaisempaa ketterän kehittämisen prosessia ovat perustana olevan prosessin käyttäminen sellaisenaan tai sen räätälöinti kokonaisuudessaan. Aikaisempaan kehittämisprosessiin perustuvan lähestymistavan vahvuudeksi todettiin helppo käyttöönotto ja integrointi olemassa oleviin menetelmiin. Prosessi ei tällöin kuitenkaan tue välttämättä riittävästi turvallisuuden suunnittelun vaatimuksia, johon prosessin räätälöinti kokonaisuudessaan todettiin vastaavan paremmin. *Yleinen prosessi* ei esitä käytettäväksi tiettyä prosessia, mutta siinä ovat mukana kehittämisen osa-alueet vaatimusmäärittelystä testaukseen sisältäen niiden turvallisuuden suunnittelun käytänteet. Yleisen prosessimallin vahvuudeksi todettiin joustava integrointi ketterän kehittämisen menetelmiin tai esimerkiksi organisaatiokohtaiseen kehittämisprosessiin. Aikaisempaan menetelmään perustuvan ja yleisen prosessin lisäksi prosessimalli voi yhdistää turvallisuuden suunnittelun ja ketterän kehittämisen käytänteitä kokonaan uudeksi prosessiksi *integroivalla prosessilla*. Lähestymistavan vahvuudeksi todettiin turvallisuuden suunnittelun käytänteiden vaatimien muutosten huomioiminen kehittämisprosessissa. Toisaalta prosessimalli nähtiin mahdollisesti liian laajana ketterän kehittämisen kannalta. Monet integrointiesitykset eivät ota kantaa käytettävään kehittämisprosessiin vaan esittävät ketterään kehittämiseen soveltuvan turvallisuuden suunnittelun käytänteen tai joukon käytänteitä. Käytänteet todettiin tehokkaiksi integroinnin apuvälineiksi, mutta tarve kehittämisprosessille tai viitekehykselle nähtiin kuitenkin tarpeelliseksi.

Turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämisen kolmeksi keskeiseksi rooliksi todettiin turvallisuusasiantuntija, kehittäjä ja asiakas. *Turvallisuusasiantuntijan* painoarvo vaihtelee integrointiesitysten välillä yhdestä turvallisuusvastaavasta useampaan turvallisuusasiantuntijaan, eikä ketterän kehittämisen rooleja esitetä välttämättä lainkaan laajennettavaksi turvallisuusasiantuntijoilla. Turvallisuusasiantuntijan tehtävänä on tyypillisesti tukea kehittämistä koko prosessin ajan turvallisuuden vaatimusmäärittelystä testaukseen. *Kehittäjillä* on integrointiesityksestä riippumatta aktiivinen rooli turvallisuuden suunnittelussa, ja he osallistuvat yleensä turvallisuustoimintojen vaatimusmäärittelyyn, toteutukseen, suunnitteluun sekä testaukseen. Ketterän kehittämisen periaatteiden mukaisesti kehittäjät tekevät yhteistyötä *asiakkaan* kanssa, joka

osallistuu turvallisuuden suunnitteluun erityisesti turvallisuustoimintojen vaatimusmäärittelyssä.

Käytänteistä ja tuotoksista yleisimmäksi todettiin käyttäjäkertomusten turvallisuuden suunnitteluun räätälöidyt versiot, jotka ovat väärinkäytön kertomukset ja turvallisuuden käyttäjäkertomukset. Hyökkääjän toimintaa luonnollisella ja epäformaalilla kielellä kuvaavat *väärinkäytön kertomukset* todettiin yksinkertaiseksi ja tehokkaaksi turvallisuuden suunnittelun käytänteeksi, joka on samalla ketterien periaatteiden mukainen. *Turvallisuuden käyttäjäkertomukset* täydentävät väärinkäytön kertomuksia huomioimalla myös tarvittavan vastatoimen, joskin vastatoimi voi olla huomioituna joissain esityksissä myös väärinkäytön kertomuksissa. Yleisellä käyttötapausten notaatiolla laadittujen *väärinkäyttötapausten* avulla voidaan muun muassa mallintaa uhkia ja määritellä niiden vastatoimet. Väärinkäyttötapausten muodollisuus nähtiin ketteryyttä vähentävänä tekijänä, mutta toisaalta sen koettiin tukevan turvallisuuden suunnittelua. *Hyökkäyspuut* muodostavat diagrammin tai hierarkkisen tekstimuotoisen kuvauksen hyökkääjän toiminnasta, ja niitä käytetään täydentämään väärinkäytön kertomuksia erityisesti monimutkaisemmissa tapauksissa. Käytänteen todettiin kattavan tarvittaessa monia hyökkäyksen variaatioita, mutta liian yksityiskohtainen mallintaminen koettiin ketteryyden periaatteiden vastaiseksi. Järjestelmän käyttäjien ja käyttöoikeuksien tunnistamisessa apuna käytettävät *roolimatriisit* auttavat esimerkiksi käyttäjäkertomusten laadinnassa. Roolimatriisit todettiin hyödylliseksi lähinnä laajemmissa järjestelmissä. *Koodaussäännöt* on johdettu ketterän kehittämisestä, ja käytänne todettiin hyväksi kehittäjien apuvälineeksi turvallisuuden suunnitteluun. *Staattisen koodin analyysin* avulla koodista voidaan löytää virheitä siihen tarkoitettujen työkalujen avulla, ja tämä puolestaan auttaa koodaussääntöjen käyttöönotossa. Ketterän kehittämisen *pariohjelmointi*, *jatkuva integrointi* ja *pienet julkaisut* todettiin turvallisuuden suunnittelua tukeviksi yksinkertaisiksi käytänteiksi. *Turvallisuustestauksen* yleisinä muotoina ovat *hyväksyntätestaus*, *yksikkötestaus* ja *läpäisytestaus*. Testaus todettiin integrointiesityksissä hyvin huomioiduksi ja toisaalta tärkeäksi osaksi muun muassa turvallisuuden varmistamista. *Turvallisuuskoulutus* koettiin hyödyllisenä, sillä se ei vaikuta kehittämisprosessiin mutta lisää yleistä turvallisuustietoisuutta.

6 YHTEENVETO JA POHDINTA

Tämän tutkielman tarkoituksena oli tarkastella turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämistä ja vastata tutkimusongelmaan: *"Miten turvallisuutta voidaan suunnitella osana ketterää järjestelmäkehitystä?"*. Tutkimusongelmaa tarkennettiin kolmella tutkimuskysymyksellä, jotka olivat:

- Millaisia menetelmiä ja standardeja on esitetty turvallisten tietojärjestelmien suunnitteluun?
- Mitä haasteita ja hyötyjä turvallisuuden suunnittelun integroimiseen ketterään ohjelmistokehitykseen liittyy?
- Millaisia menetelmiä ja käytänteitä on esitetty turvallisuuden suunnitteluun osana ketterää kehittämistä?

Ensimmäisen tutkimuskysymyksen tarkoituksena oli muodostaa turvallisuuden suunnittelun kohdealueen perusta. Tutkimuskysymykseen vastaamiseksi esiteltiin keskeiset turvallisuuden suunnittelun käsitteet useisiin standardeihin perustuen ja tarkasteltiin laajemmin kahta yleistä tietoturvastandardia, niiden sovelluksia sekä kahta turvallisuuden suunnittelun käytännettä. Toisen tutkimuskysymyksen perustaksi tarkasteltiin, mitä ketterällä ohjelmistokehityksellä tarkoitetaan, ja esiteltiin kahta yleisintä ketterää menetelmää, XP:tä ja Scrumia. Tutkimuskysymykseen vastaamiseksi esiteltiin kirjallisuudessa esitettyjä integroinnin haasteita ja hyötyjä. Kolmanteen tutkimuskysymykseen vastaamiseksi esiteltiin useita integrointiin esitettyjä menetelmiä ja käytänteitä sekä vertailtiin niiden piirteitä tutkimuksen viitekehyksen avulla.

Relevanttien julkaisujen hakemisessa ja valinnassa sovellettiin integroivan kirjallisuuskatsauksen menetelmää (Torraco, 2005). Prosessin tuloksena tuli valituksi yhdeksän julkaisua, joissa tarkastellaan turvallisuuden suunnittelua ketterän ohjelmistokehityksen yhteydessä.

Tutkielman kirjallisuuskatsauksen perusteella turvallisuusstandardit ja turvallisuuden suunnittelun menetelmät todettiin kattaviksi muun muassa järjestelmien turvallisuustason arviointiin ja turvallisuuden suunnitteluun, mutta ne koettiin sellaisenaan huonosti soveltuviksi ohjelmisto- ja järjestelmäkehityk-

seen. Ketterän kehittämisen ja turvallisuuden suunnittelun yleiseksi haasteeksi todettiin eroavaisuudet näiden kahden osa-alueen *prosessissa*. Haasteellisena nähtiin myös *dokumentaatio* ja sen avulla osoitettava *turvallisuuden varmistaminen*. Ketterien menetelmien yleiseksi puutteeksi todettiin turvallisuuden suunnittelun näkökulmasta myös *keskittyminen toiminnallisiin vaatimuksiin*. Toisaalta ketterän kehittämisen turvallisuuden suunnittelua tukeviksi piirteiksi todettiin *kommunikointi*, *reagointi ympäristön muutoksiin* sekä *turvallisuuspuutteiden asteittaisen löytäminen*.

Ketterän kehittämisen ja turvallisuuden suunnittelun integraation todettiin olevan kompromissi, jonka painopisteen määrittää lopulta integrointiin käytetyn menetelmän piirteet. Arviointia ja vertailua varten luotiin viitekehys, jonka avulla integrointiesitysten yksittäisten piirteiden painotusta turvallisuuteen tai ketteryyteen voitiin jäsennellä tarkemmin. Viitekehyksen neljä osaa ovat *prosessi*, *roolit*, *käytänteet* ja *tuotokset*. Ketterän kehittämisen ja turvallisuuden suunnittelun integroinnin prosessi perustuu yleisesti *aikaisempaan ketterän kehittämisen menetelmään*, kuten XP:hen tai Scrumiin. Muita prosessimalleja ovat *yleinen*, joka kattaa kehittämisen osa-alueet yleiskäyttöisellä prosessilla, sekä *integroiva*, joka esittää uuden integrointia tukevan prosessin. Monet integrointiesitykset eivät esitä lainkaan prosessia, vaan tarkastelevat integrointia ainoastaan käytänteiden kannalta. Prosessin näkökulmasta integrointi painottuu joka tapauksessa ketterään kehittämiseen, mikä on luonnollista ottaen huomioon turvallisuuden suunnittelumenetelmien piirteet; jos kehittämisprosessia laajennetaan merkittävästi, ei kyseessä ole enää ketterä menetelmä.

Integrointiesityksistä on tunnistettavissa kolme keskeistä roolia, jotka ovat turvallisuusasiantuntija, kehittäjä ja asiakas. *Turvallisuusasiantuntijan* rooli nähtiin ketterien periaatteiden mukaisesti enimmäkseen kehitystyötä tukevana. *Kehittäjät* osallistuvat aktiivisesti turvallisuuden suunnitteluun käytännössä kaikissa vaiheissa turvallisuustoimintojen vaatimusmäärittelystä testaukseen. *Asiakkaan* keskeisimmäksi turvallisuuden suunnittelun osa-alueeksi todettiin turvallisuustoimintojen vaatimusmäärittely, johon asiakas tuo lisäarvoa liiketoiminnan ja suojattavien kohteiden tuntemuksella. Yleisesti ottaen roolit ovat yhteneväisiä ketterän kehittämisen tavallisiin rooleihin, mutta tehtävät luonnollisesti laajenevat turvallisuuden suunnittelun käytänteillä.

Käytänteet ja niiden tuotokset ovat keskeinen osa turvallisuuden suunnittelun ja ketterän kehittämisen yhdistämistä. Yleisimpiä käytänteitä ovat *väärinkäytön kertomukset* ja *turvallisuuden käyttäjäkertomukset*, joiden todettiin tukevan turvallisuuden suunnittelua useisiin tilanteisiin riittävällä tasolla muun muassa järjestelmän riskien ja haavoittuvuuksien arvioinnissa sekä turvallisuuden varmistamisessa. *Väärinkäyttötapauksia* hyödynnetään yleisesti turvallisuuden suunnittelun käytänteenä, ja sen väärinkäytön kertomuksia muodollisempi notaatio koettiin turvallisuuden suunnittelua tukevaksi. Järjestelmän väärinkäyttöä kuvataan myös *hyökkäyspuilla*, joiden tarkkuustason ja mallintamisen laajuuden todettiin kuitenkin vaativan harkintaa ketteryyden näkökulmasta. Kaikissa käytänteissä turvallisuuden suunnittelua ei lähestytä hyökkääjän kannalta. *Roolimatriisi* auttaa järjestelmän käyttäjien ja erityisesti pääsyoikeuksien tunnis-

tamisessa. *Koodaussäännöt* ja *staattisen koodin analyysi* auttavat kehittäjiä turvallisen koodin tuottamisessa koko kehitystyön ajan. *Pariohjelmointi*, *jatkuva integrointi* ja *pienet julkaisut* todettiin yksinkertaisiksi ketterän kehittämisen käytänteiksi, joilla voidaan tukea turvallisuuden suunnittelua. *Turvallisuustestaus* on huomioitu eri muodoissa useimmissa integrointiesityksissä, mikä on tärkeää muun muassa turvallisuuden varmistamisen kannalta. Kehittäjien ja sidosryhmien yleinen turvallisuustietoisuus voidaan huomioida *turvallisuuskoulutuksella*, joka ei myöskään vaikuta muihin käytänteisiin ja on siten helppo toteuttaa. Edellä mainitut yleisimmät käytänteet muodostavat kattavan läpileikkauksen niin ketterän kehittämisen kuin turvallisuuden suunnittelunkin käytänteistä. Osa käytänteistä, kuten hyökkäyspuut tai roolimatriisi, on johdettu suoraan turvallisuuden suunnittelusta, kun taas pariohjelmointi, jatkuva integrointi ja pienet julkaisut ovat esimerkkejä ketterän kehittämisen sellaisenaan hyödynnettävistä käytänteistä. Väärinkäytön kertomukset puolestaan yhdistävät molemmat osa-alueet.

Tässä tutkielmassa esitettyjä ratkaisuehdotuksia voidaan hyödyntää monipuolisesti eri käyttötarkoituksiin niin turvallisuuskriittisissä ohjelmistoprojekteissa kuin yleisesti ketterässä järjestelmäkehityksessä. Esitetyt menetelmät ja käytänteet osoittavat ratkaisun useimpiin integroinnin haasteisiin, kuten prosessien eroihin, turvallisuuden ei-toiminnallisten vaatimusten toteuttamiseen tai turvallisuuden varmistamiseen. Integrointiesitykset keskittyvät yleisesti haasteiden ratkaisuehdotuksiin, mutta ei ole syytä unohtaa myöskään ketterän kehittämisen turvallisuuden suunnittelua tukevia piirteitä, kuten kommunikointia, turvallisuuspuutteiden asteittaista löytämistä tai reagointia ympäristön muutoksiin. Nämä tukevat turvallisuuden suunnittelun sosiaalista ja sosioteknistä näkökulmaa (vrt. McGraw, 2004), jotka on osaltaan huomioitu myös tutkielman ratkaisuehdotuksissa. Yleisenä johtopäätöksenä voidaan todeta, että ketterä kehittäminen ei ole ristiriidassa turvallisuuden suunnittelun kanssa. On kuitenkin selvää, että esimerkiksi uhkien mallinnus ja riskien tunnistaminen sekä tuotettavat artefaktit ovat integrointiratkaisuissa turvallisuuden suunnittelun menetelmiä suppeampia. Integroinnin ratkaisuehdotukset keskittyvät myös yleisesti tiettyyn kehittämisen osa-alueeseen, kuten vaatimusmäärittelyyn, jolloin kokonaisuus voi jäädä puutteelliseksi. Kehittämisprosessia esittäviä ratkaisuehdotuksia, joissa myös käytänteitä on pohdittu monipuolisesti, on julkaistu kirjallisuuskatsauksen perusteella verrattain vähän.

Turvallisuuden suunnittelu ja ketterä kehittäminen ovat tutkimusalueena laajoja. Vaikka tutkielman perustana käytettiin kattavasti näiden kahden osa-alueen integrointia käsittelevää lähdeaineistoa, jouduttiin osa käytänteistä ja yleisestä aihepiirin kirjallisuudesta rajaamaan tarkastelun ulkopuolelle. Lisäksi turvallisuusstandardit ovat erittäin laajoja, jonka vuoksi yksityiskohtaisempaan tarkasteluun valittiin vain kaksi yleistä standardia. Rajausten vuoksi aihepiirin tarkastelu perustuu osittain yksittäisiin lähdeaineistoihin. Koska tutkielman turvallisuuden suunnittelun viitteenä hyödynnetyt turvallisuusstandardit osoittavat kuitenkin yleisesti samoja turvallisuuden suunnittelun osa-alueita ja käsiteltyjen integrointiesitysten määrää voidaan pitää kattavana, on tutkimusase-

telma mahdollistanut kuitenkin hyvin tutkimusongelman kannalta keskeiset johtopäätökset.

Aihepiirin aiempia kirjallisuuskatsauksia on tehty suppeasti osana useimpia integrointiesityksiä, mutta ne eivät ole kuitenkaan suoraan verrattavissa tämän tutkielman kokoavaan esitykseen. Aikaisemmasta aineistosta vertailukohtana voidaan pitää Bartschin (2011) esitystä, joka kokoaa kirjallisuudessa mainittuja integroinnin haasteita ja esitettyjä ratkaisuja. Julkaisussa on esitetty osittain samoja haasteita kuin tässä tutkielmassa, kuten prosessin eroavaisuudet, turvallisuuden varmistaminen ja keskittyminen toiminnallisiin vaatimuksiin. Myös ratkaisuehdotukset sisältävät tässä tutkielmassa esitettyjä käytänteitä, kuten väärinkäyttötapaukset, asteittainen turvallisuuden suunnittelu ja turvallisuustestaus. Bartschin (2011) kirjallisuuskatsaus on kuitenkin hieman suppeampi, ja tutkimuksen pääpaino on asiantuntijahaastatteluissa. Kirjallisuuskatsauksen analyysin tulokset ovat kuitenkin yhteneväisiä tämän tutkielman kanssa. Sen sijaan Bartschin (2011) asiantuntijahaastattelujen perusteella turvallisuuden asiantuntemus tiimissä tai väärinkäyttötapaukset eivät ole keskeisiä integroinnin kannalta. Näkemys on osittain ristiriidassa yleiseen aihepiirin kirjallisuuteen ja tämän tutkielman johtopäätöksiin, mutta toisaalta Bartschin (2011) tulokset perustuvat vain kymmenen asiantuntijan haastatteluun, mitä voidaan pitää melko suppeana.

Vaikka tutkielman tuloksena tuotettiin kokoava katsaus turvallisuuden suunnittelun ja ketterän kehittämisen integroinnin ratkaisuehdotuksista, ei tutkielmassa otettu kantaa kehitystyön jälkeiseen aikaan. Kun tarkastellaan minkä tahansa järjestelmän elinkaarta, on turvallisen järjestelmän kehittäminen vain yksi osa-alue turvallisuuden kokonaisuutta. Haavoittuvuuksilta suojautuminen ketterällä ja ennaltaehkäisevällä tavalla järjestelmän käyttöönoton jälkeen on yksi mahdollinen jatkotutkimusaihe. Myös integrointiin esitettyjen ratkaisuehdotusten empiiriselle tarkastelulle on hyvin tilaa, sillä aikaisempi aineisto perustuu enimmäkseen teoreettiseen tietoon. Osittain irrallisten ratkaisuehdotusten kokoaminen samaan viitekehykseen empiirisen tiedon tukemana mahdollistaisi tutkimusalueen kehittämisen askeleen eteenpäin.

LÄHTEET

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods. Review and Analysis.* VTT Publications, 478. Technical Research Centre of Finland

Agile Alliance. (2001). *Manifesto for Agile Software Development*. Haettu 26.3.2014 osoitteesta http://www.agilemanifesto.org/

Alexander, I. (2002). Initial industrial experience of misuse cases in trade-off analysis. Teoksessa *IEEE Joint International Conference on Requirements Engineering* (s. 61-68). Los Alamitos, CA: IEEE Computer Society.

Ambler, S. (2002). *Agile modeling: effective practices for extreme programming and the unified process*. New York : John Wiley & Sons.

Amoroso, E. (1994). *Fundamentals of computer security technology*. Englewood Cliffs, N.J.: PTR Prentice Hall.

Anderson, D. (2010). *Kanban: Succesful Evolutionary Change for Your Technology Business*. Washington: Blue Hole Press.

Anderson, J. P. (1972). *Computer Security Technology Planning Study. Volume 2*. Fort Washington, PA : James P. Anderson & Co.

Aydal, E. G., Paige, R. F., Chivers, H., & Brooke, P. J. (2006). Security planning and refactoring in extreme programming. Teoksessa P. Abrahamsson, M. Marchesi & G. Succi (toim.), *Extreme Programming and Agile Processes in Software Engineering* (s. 154-163). Berlin : Springer Heidelberg.

Azham, Z., Ghani, I., & Ithnin, N. (2011). Security backlog in Scrum security practices. Teoksessa M. F. Harun & A. Selamat (toim.) *2011 5th Malaysian Conference in Software Engineering (MySEC), 13-14 December, 2011, Johor Bahru, Malaysia* (s. 414-417). IEEE Computer Society.

Baca, D., & Petersen, K. (2010). Prioritizing countermeasures through the countermeasure method for software security (CM-Sec). Teoksessa M. Ali Babar, M. Vierimaa & M. Oivo (toim.) *Product-Focused Software Process Improvement* (s. 176-190). Berlin : Springer Heidelberg.

Baca, D., & Carlsson, B. (2011). Agile development with security engineering activities. Teoksessa *Proceedings of the 2011 International Conference on Software and Systems Process* (s. 149-158). New York, NY: ACM Press.

Bansal, S. K., & Jolly, A. (2014). An encyclopedic approach for realization of security activities with agile methodologies. Teoksessa *Proceedings of the 5th International Conference, Confluence 2014, The Next Generation Information Technology Summit* (s. 767-772). IEEE Computer Society.

Bartsch, S. (2011). Practitioners' Perspectives on Security in Agile Development. Teoksessa *2011 Sixth International Conference on Availability, Reliability and Security (ARES)* (s. 479-484). IEEE Computer Society.

Baskerville, R. (1992). The developmental duality of information systems security. *Journal of Management Systems*, 4(1), 1-12.

Beck, K. & Andres, C. (2004). *Extreme programming explained: embrace change*. (2.painos). Addison-Wesley Professional.

Beck, K. (1999a). Embracing change with extreme programming. *Computer*, 32(10), 70-77.

Beck, K. (1999b). *Extreme Programming Explained*. (1. painos). Addison–Wesley Professional.

Beznosov, K. (2003). Extreme security engineering: on employing XP practices to achieve'good enough security'without defining it. Teoksessa *First ACM Workshop on Business Driven Security Engineering (BizSec), Fairfax, VA (Vol. 31)*. CiteSeerX.

Beznosov, K., & Kruchten, P. (2004). Towards agile security assurance. Teoksessa *Proceedings of the 2004 Workshop on New Security Paradigms* (s. 47-54). New York, NY: ACM Press.

Boehm, B., & Turner, R. (2003). Observations on balancing discipline and agility. Teoksessa *Proceedings of the Agile Development Conference, 2003* (s. 32-39). Los Alamitos, CA: IEEE Computer Society.

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., & Kruchten, P. (2006). Extending XP practices to support security requirements engineering. Teoksessa *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems* (s. 11-18). New York, NY: ACM Press.

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *Software*, IEEE, 25(1), 60-67.

CC. (2012). *Common Criteria for Information Terchnology Security Evaluation. Version 3.1, revision 4*. Haettu 1.4.2014 osoitteesta http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4_marked_changes.pdf

Chan, M. T., & Kwok, L. F. (2001). Integrating security design into the software development process for e-commerce systems. *Information Management & Computer Security*, 9(3), 112-122.

Chivers, H., Paige, R. F., & Ge, X. (2005). Agile security using an incremental security architecture. Teoksessa H. Baumeister, M. Marchesi & M. Holcombe (toim.) *Extreme Programming and Agile Processes in Software Engineering* (s. 57-65). Berlin : Springer Heidelberg.

Chung, L., & Nixon, B. A. (1995). Dealing with non-functional requirements: three experimental studies of a process-oriented approach. Teoksessa *17th International Conference on Software Engineering, (ICSE 1995)* (s. 25-25). IEEE Computer Society.

Conboy, K. (2009). Agility from first principles: reconstructing the concept of agility in information systems development. *Information Systems Research,* 20(3), 329-354.

Cockburn, A. (2002). *Agile software development*. Boston : Addison-Wesley.

Cockburn, A. & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131-133.

Cooper, H. M. (1998). *Synthesizing research: a guide for literature reviews* (2.painos). Thousand Oaks : Sage.

Davis, N. (2005). *Secure software development life cycle processes: a technology scouting report (No. CMU/SEI-2005-TN-024)*. Haettu 1.4.2014 osoitteesta www.sei.cmu.edu/reports/05tn024.pdf

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9), 833-859.

Farroha, D., & Farroha, B. (2011, November). Developing corporate services in an agile environment. Teoksessa *Military Communications Conference, (MILCOM 2011)* (s. 1535-1540). IEEE Computer Society.

Firesmith, D. G. (2003). Security use cases. *Journal of Object Technology*, 2(3).

Ge, X., Paige, R. F., Polack, F. A., Chivers, H., & Brooke, P. J. (2006). Agile development of secure web applications. Teoksessa *Proceedings of the 6th International Conference on Web Engineering* (s. 305-312). New York, NY: ACM Press.

Ge, X., Paige, R. F., Polack, F., & Brooke, P. (2007). Extreme programming security practices. Teoksessa G. Concas, E. Damiani, M. Scotto & G. Succi (toim.) *Agile Processes in Software Engineering and Extreme Programming* (s. 226-230). Berlin : Springer Heidelberg.

Goertzel, K. M., Winograd, T., McKinley, H. L., Oh, L. J., Colon, M., McGibbon, T., & Vienneau, R. (2007). *Software Security Assurance: a State-of-Art Report (SAR).* Haettu 1.4.2014 osoitteesta http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA472363

Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley.

Hoffman, L. J. (1977). *Modern methods for computer security and privacy*. Englewood Cliffs: Prentice-Hall.

Howard, M., & Lipner, S. (2009). *The security development lifecycle*. New York : O'Reilly Media, Incorporated.

IEEE 1228. (1994). *IEEE Standard for Software Safety Plans. (Std IEEE1228-1994)* (s. 1-23). IEEE Computer Society.

ISO/IEC. (2009). *Guide 73*. Haettu 4.5.2014 osoitteesta https://www.iso.org/obp/ui/#iso:std:iso:guide:73:ed-1:v1:en:term:3.6.1.1

ISO/IEC 15408. (2008a). *Information technology – Security techniques – Evaluation criteria for IT security – Part2: Security functional components (Std ISO IEC 15408-2 – 2008)* (s. 1-240). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 15408. (2008b). *Information technology – Security techniques – Evaluation criteria for IT security – Part3: Security assurance components (Std ISO IEC 15408-3 – 2008)* (s. 1-188). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 21827. (2008). *Information technology – Security techniques – Systems Security Engineering – Capability Maturity Model (SSE-CMM) (Std ISO IEC 21827 – 2008)* (s. 1-154). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 17799. (2005). *Information technology – Security techniques – Code of practice for information security management (Std ISO IEC 17799 – 2005)* (s. 1-126). Geneva, Switzerland: International Organization for Standardization.

ITIL. (2011). *An Introductory Overview of ITIL 2011. Best Management Portfolio Product*. Haettu 4.5.2014 osoitteesta https://www.best-management-practice.com

Jacobson I., Booch G. & Rumbaugh J. (1999). *The Unified Software Development Process*. Addison-Wesley.

Kankanhalli, A., Teo, H. H., Tan, B. C., & Wei, K. K. (2003). An integrative study of information systems security effectiveness. *International Journal of Information Management*, 23(2), 139-154.

Keramati, H., & Mirian-Hosseinabadi, S. (2008, March). Integrating software development security activities with agile methodologies. Teoksessa *International Conference on Computer Systems and Applications*, 2008 (s. 749-754). IEEE Computer Society.

Kniberg, H. (2007). *Scrum and XP from the Trenches*. Haettu 1.4.2014 osoitteesta http://www.infoq.com/minibooks/scrum-xp-from-the-trenches

Kulak, D., & Guiney, E. (2004). *Use cases: requirements in context*. Addison-Wesley Professional.

Kongsli, V. (2006). Towards agile security in web applications. Teoksessa *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications* (s. 805-808). New York, NY: ACM Press.

Laanti, M. (2013). *Agile methods in large-scale software development organizations – applicability and model for adoption*. Tampere : Juvens Print.

Leffingwell, D., Jemilo, D., Zamora, M., O´Neill, C., Yakuma, A. (2014). *The Scaled Agile Framework, (SAFE)*. Haettu 27.11.2014 osoitteesta http://scaledagileframework.com/

Lee, Y., Lee, J., & Lee, Z. (2002). Integrating software lifecycle process standards with security engineering. *Computers & Security*, 21(4), 345-355.

Ma, Q., Johnston, A. C., & Pearson, J. M. (2008). Information security management objectives and practices: a parsimonious framework. *Information Management & Computer Security*, 16(3), 251-270.

Mauw, S., & Oostdijk, M. (2006). Foundations of attack trees. Teoksessa D. H. Won & S. Kim (toim.) *Information Security and Cryptology-ICISC 2005* (s. 186-198). Berlin : Springer Heidelberg.

McCormick, M. (2001). Technical opinion: Programming extremism. *Communications of the ACM*, 44(6), 109-119.

McDermott, J., & Fox, C. (1999). Using abuse case models for security requirements analysis. Teoksessa *15th Annual Computer Security Applications Conference, (ACSAC'99)* (s. 55-64). Los Alamitos, CA: IEEE Computer Society.

McGraw, G. (2004). Software security. *Security & Privacy, IEEE,* 2(2), 80-83.

McGraw, G. (2006). *Software security: building security in* (1.painos). Upper Saddle River, NJ : Addison-Wesley Professional.

Mellado, D., Fernández-Medina, E., & Piattini, M. (2007). A common criteria based security requirements engineering process for the development of secure information systems. *Computer Standards & Interfaces*, 29(2), 244-253.

Mougouei, D., Sani, N. F. M., & Almasi, M. M. (2013). S-Scrum: a Secure Methodology for Agile Development of Web Services. *World of Computer Science and Information Technology Journal, (WCSIT)*, 3(1), 15-19.

Mouratidis, H., Giorgini, P., & Manson, G. (2003). Integrating security and systems engineering: Towards the modelling of secure information systems. Teoksessa J. Eder & M. Missikoff (toim.) *Advanced Information Systems Engineering* (s. 63-78). Berlin : Springer Heidelberg.

Musa, S. B., Norwawi, N. M., Selamat, M. H., & Sharif, K. Y. (2011, March). Improved extreme programming methodology with inbuilt security. Teoksessa *2011 IEEE Symposium on Computers & Informatics, (ISCI)* (s. 674-679). IEEE Computer Society.

Othmane, L., Angin, P., Weffers, H., & Bhargava, B. (2013). Extending the Agile Development Approach to Develop Acceptably Secure Software. Teoksessa *IEEE Transactions on Dependable and Secure Computing (s.* 497 – 509). IEEE Computer Society.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using scrum in distributed agile development: A multiple case study. Teoksessa *Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009)* (s. 195-204). Los Alamitos, CA: IEEE Computer Society.

Palmer, S. R., & Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.

Parker, D. B., Nycum, S., & Oüra, S. S. (1973). *Computer abuse*. California: Stanford Research Institute.

Parnas, D. and Clements, P. (1986). A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering*. 19(2), 251-257.

Paulk, M. C. (2001). Extreme programming from a CMM perspective. *Software, IEEE*, 18(6), 19-26.

Pearson, S., & Yee, G. (2013). *Privacy and Security for Cloud Computing*. Berlin: Springer Heidelberg.

Peeters, J. (2005). *Agile security requirements engineering*. Haettu 1.4.2014 osoitteesta http://www.johanpeeters.com/papers/abuser%20stories.pdf

Phillips, C., & Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. Teoksessa *Proceedings of the 1998 Workshop on New Security Paradigms* (s. 71-79). New York, NY: ACM Press

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Boston : Addison-Wesley Professional.

Posthumus, S., & Von Solms, R. (2004). A framework for the governance of information security. *Computers & Security*, 23(8), 638-646.

Rodríguez, P., Markkula, J., Oivo, M., & Turula, K. (2012). Survey on agile and lean usage in finnish software industry. Teoksessa *Proceedings of the ACM-*

IEEE International Symposium on Empirical Software Engineering and Measurement (s. 139-148). New York, NY: ACM Press.

Royce, W. (1970). Managing the development of large software system: Concepts and techniques. Teoksessa *Proceedings of the 9th International Conference on Software Engineering* (s. 1–9). IEEE Computer Society.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Haettu 1.4.2014 osoitteesta http://msdl.cs.mcgill.ca/people/tfeng/docs/The%20Unified%20Modeling%20Language%20Reference%20Manual.pdf

Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *Software, IET,* 2(1), 58-64.

Sindre, G., & Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requirements Engineering,* 10(1), 34-44.

Singhal S. & Singhal A. (2011). Development of agile security framework using a hydrid technique for requirements elicitation. Teoksessa S. Unnikrishnan, S. Surve & D. Bhoir (toim.) *Advances in Computing, Communication and Control, LNCS 125* (s. 178-188). Berlin: Springer-Heidelberg.

Schneier, B. (1999). Attack trees. *Dr. Dobb's journal,* 24(12), 21-29.

Schwaber, K. & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice-Hall.

Schwaber, K. (2004). *Agile project management with Scrum (Vol. 7)*. Redmond: Microsoft press.

Siponen, M., & Baskerville, R. (2001). A new paradigm for adding security into IS development methods. Teoksessa J. H. P. Eloff, L. Labuschagne, R. von Solms & G. Dhillon (toim.) *Advances in Information Security Management & Small Systems Security* (s. 99-111). Springer US.

Siponen, M., Baskerville, R., & Kuivalainen, T. (2005). Integrating security into agile development methods. Teoksessa *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (s. 185a) . IEEE Computer Society.

Siponen, M. (2005). An analysis of the traditional IS security approaches: implications for research and practice. *European Journal of Information Systems*, 14(3), 303-315.

Stoneburner, G., Goguen, A. & Feringa, A. (2002). Risk management guide for information technology systems. *Nist special publication*, 800(30), 800-30.

Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007, January). Distributed scrum: Agile project management with outsourced development teams. Teoksessa *Proceedings of the Fortieth Annual Hawaii International Conference on System Sciences* (s. 274a). Los Alamitos, CA: IEEE Computer Society.

Sutherland, J. & Schwaber, K. (2013). *The Scrum Guide*. Haettu 21.8.2014 osoitteesta https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-FI.pdf

Stapleton, J. (1997). *DSDM, dynamic systems development method*. Harlow, England: Addison-Wesley.

Stevens, D. (2011). *What is next for the Agile Manifesto*. Haettu 21.8.2014 osoitteesta http://www.dennisstevens.com/2011/02/13/whats-next-for-the-agile-manifesto/

Stålhane, T., Myklebust, T., & Hanssen, G. K. (2012). *The application of Safe Scrum to IEC 61508 certifiable software*. Haettu 1.4.2014 osoitteesta https://www.sintef.no/globalassets/upload/final-psam-11-esrel-2012.pdf

Tappenden, A., Beatty, P., Miller, J., Geras, A., & Smith, M. (2005). Agile security testing of web-based systems via httpunit. Teoksessa *Proceedings of Agile Conference 2005, 24-29 July* (s. 29-38). Los Alamitos, CA: IEEE Computer Society.

Torraco, R. J. (2005). Writing integrative literature reviews: Guidelines and examples. *Human Resource Development Review*, 4(3), 356-367.

VersionOne. (2013). *8th Annual State of Agile Development Survey*. Haettu 5.8.2014 osoitteesta http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

Vetterling, M., Wimmel, G., & Wisspeintner, A. (2002). Secure systems development based on the common criteria: the PalME project. *ACM SIGSOFT Software Engineering Notes*, 27(6), 129-138.

Wäyrynen J., Boden M. & Boström G. (2004). Security engineering and eXtreme Programing: an impossible marriage? Teoksessa C. Zannier, H. Erdogmus & L. Lindstrom (toim.) *Extreme Programming and Agile Methods - XP/Agile Universe 2004, LNCS 3134*, (s. 117-128). Berlin: Springer Heidelberg.