Santtu Rinta-Nikkola

Kanban ohjelmistokehityksessä: Case tietokonepelin kehitys

Tekniikan ja innovaatiojohtami-
sen akateeminen yksikkö
Kauppatieteen pro gradu
Tietojärjestelmätiede

Vaasa 2021

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen akateeminen yksikkö
Tekijä: Santtu Rinta-Nikkola
Tutkielman nimi: Kanban ohjelmistokehityksessä: Case tietokonepelin kehitys
Tutkinto: Kauppatieteiden maisteri
Oppiaine: Tietojärjestelmätiede
Työn ohjaaja: Teemu Mäenpää
Valmistumisvuosi: 2021 **Sivumäärä:** 91

TIIVISTELMÄ:
Tässä pro gradu -tutkielmassa tutkittiin Kanbanin soveltuvuutta ketteränä ohjelmistokehityksen projektinhallintamenetelmänä. Ohjelmistokehityksessä ketterien projektinhallintamenetelmien tarve on kasvanut nopeasti ja oikean projektinhallintamenetelmän valinta on kriittinen osa modernia ohjelmistokehityksen prosessia. Tutkielmassa myös käsiteltiin projektinhallinnan historiaa ja sitä miksi perinteiset projektinhallinnantyökalut eivät välttämättä ole riittäviä ohjelmistokehityksessä.

Kanbanista tehtyjä ohjelmistokehityksen tutkimuksia on hyvin niukasti ja tutkimuksen puute saattaa johtaa Kanbanin ohittamiseen projektinhallintamenetelmän valinnassa. Tämä tutkielma pyrkii lisäämään Kanbanista löydettävien tietojen ja tutkimuksien määrää vastaamalla tutkimuskysymykseen: ”Kuinka Kanban soveltuu ketteränä projektinhallintamenetelmänä ohjelmistokehitykseen?”

Tutkielman tavoite pyrittiin saavuttamaan tarkastelemalla kattavasti ensin tutkielman aiheeseen liittyviä tutkimuksia sekä niihin liittyvää kirjallisuutta. Tämän jälkeen tutkielman projektin tekoa varten luotiin aikataulu ja valittiin ohjelmistokehitystä varten soveltuvat ohjelmistot. Ohjelmistojen ja kirjallisuuden kattavan tarkastelun jälkeen aloitettiin projektin työstäminen. Työn tekemistä johti tietokonepelin kehitys Kanban -projektinhallintamenetelmää käyttäen. Tutkielman tapaus suoritettiin käyttämällä Kanbania itsenäisen pelinkehitysprojektin hallintamenetelmänä.

Tutkielman tutkimusmenetelmänä oli tapaustutkimus, jossa tarkasteltiin Kanbanin soveltuvuutta ohjelmistokehitykseen ketteränä projektinhallintamenetelmänä tuottamalla itsenäinen pelinkehitysprojekti. Tapaustutkimuksesta saatuja tuloksia analysoitiin fenomenologisella analyysilla. Fenomenologista analyysia varten tutkimuksessa saadut havainnot kirjattiin muistikirjaan sekä kuva kaapattiin.

Tutkielmassa esille tulleet löydökset tukivat jo olemassa olevaa teoriaa Kanbanin käytöstä ohjelmistokehityksessä. Huomattavin poikkeus jo olemassa olevien tutkimuksien tuloksiin tuli Kanbanin soveltuvuudesta itsenäiseen kehitystyöprojektiin. Tutkielman tuloksissa Kanbanin toiminta itsenäisessä kehitystyöprojektissa oli riittävä, kun taas suurempien projektien tuloksissa Kanban ei yksinään riitä hallitsemaan koko projektin kulkua. Tämän takia Kanban on tehokkaasti toimiva työkalu itsenäiseen kehitystyöhön, mutta suuremmissa projekteissa Kanban toimii projektin visualisoijana sekä Juuri Oikeaan Tarpeeseen (JOT) -järjestelmän ohjaustyökaluna.

AVAINSANAT: Projektinhallinta, ketterä, Scrum, Kanban, Scrumban, Lean-ajattelu, ohjelmistokehitys, pelinkehitys.

Sisällys

Kuvat

Kuviot

Ohjelmat

Taulukot

Määritelmät ja Lyhenteet

Agile Project Management (APM)	Ketterä projektinhallinta.
Application Programming Interface (API)	Ohjelmointirajapinta eli yhtymäkohta, joka mahdollistaa tiedon siirron kahden tai useamman ohjelman välillä.
Critical Path Method (CPM)	Kriittisen polun menetelmä.
Game as a Service (GaaS)	Pelin tarjoaminen palveluna.
Indie-videopeli	Videopeli, joka on kehitetty ilman julkaisijan antamaa taloudellista tukea. Indie tulee englanninkielisestä sanasta *independent,* eli omatoiminen.
Mikrotransaktio	Pelinsisäinen mikromaksu.
Monetisointi	Tuotteen tai palvelun rahaksi lyöminen.
Photoshop Document (.PSD)	Adobe Photoshop tiedostomuoto.
Software as a Service (SaaS)	Ohjelmiston tarjoaminen palveluna.
Tekninen velka	Ohjelmistokehityksessä kasaantunut velka, joka yleensä johtuu nopeasti ja hätäisesti tehdyistä ratkaisuista. Tekninen velka aiheuttaa ongelmia ohjelmien ylläpidossa ja jatkokehityksessä.
Tietue	Tietojenkäsittelyssä esiintyvä tietorakenne.
Traditional Project Management (TPM)	Perinteinen projektinhallinta.

1 Johdanto

Tässä pro gradu -tutkielmassa tarkastellaan ja tutkitaan Kanbanin soveltuvuutta ketteränä projektinhallintamenetelmänä ohjelmistokehityksessä. Tutkimusta varten tutkielmassa tarkastellaan ensin projektinhallinnan merkitystä sekä erilaisten projektinhallintakäytäntöjen eroavaisuuksia. Kanban on ketterä projektinhallintamenetelmä, mutta Kanban ei käytä ketterän ohjelmistokehityksen julistuksessa määriteltyä toistuvaa lähestymistapaa. Kanbanissa projektin työvirta on jatkuva eli Kanban mukautuu muutoksiin heti niiden ilmentyessä (Kanbanize, 2020a). Kanban on ketterien projektinhallintamenetelmien tapaan joustava menetelmä, joka pystyy mukautumaan erilaisten projektien tarpeisiin. Ketterät projektinhallintamenetelmät ovat ohjeita antavia viitekehyksiä ja tämän takia ketterien menetelmien käyttö projekteissa on projektikohtaista. Kanbanin jatkuva työvirta soveltuu hyvin muutosalttiisiin projekteihin ja tämän takia Kanban voi olla tehokas valinta ohjelmistokehityksen hallintamenetelmän valinnaksi.

Kanbanin soveltuvuudesta ohjelmistokehityksen ketteränä projektinhallintamenetelmänä on löytyvissä hyvin vähän edeltävää tutkimusta. Ahmadin, Markkulan ja Oivon (2013) tuottaman kirjallisuuskatsauksen mukaan Kanbanista on tehty vain yhdeksäntoista tutkimusta vuodesta 2000 vuoteen 2011. Vähäisen tutkimuksen määrä Kanbanin toimivuudesta mahdollistaa useiden eri lisätutkimuksien luomisen Kanbanin käytöstä ohjelmistokehityksessä.

Tutkielman tapauksen aiheena on luoda Kanbania käyttävä itsenäinen pelinkehitysprojekti. Itsenäinen pelinkehitystyö on Kanbanin lisäksi aihealue, josta jo olemassa olevan tutkimuksen määrä on hyvin vähäistä. Pelinkehitys on osa ohjelmistokehitystä ja molemmat kehitysalueet jakavat samankaltaiset kehitysprosessit (Bethke, 2003). Itsenäisessä pelinkehityksessä Kanbanin käytöstä saatavat tulokset voidaan tämän perusteella vertailla ohjelmistokehityksen tuloksiin. Tulosten vertailu tapahtuu käyttämällä fenomenologista analyysia, jossa tutkimuksesta saadut havainnot ja kokemukset verrataan jo olemassa oleviin tutkimuksiin. Ketterien projektinhallintamenetelmien pääperiaate on

tarjota kehitystyöhön joustavat työkalut, joiden avulla tuetaan mahdollisuutta vastata muutoksiin sekä vahvistetaan yksilöiden ja asiakkaiden yhteistyötä.

Tutkielmassa tuotettava itsenäisen pelinkehityksen tapaus Kanbania käyttämällä pyrkii vastaamaan tutkimuskysymykseen:

- *Kuinka Kanban soveltuu ketteränä projektinhallintamenetelmänä ohjelmistokehitykseen?*

Tutkielman tapausta varten tutkielmassa tarkastellaan huolellisesti projektinhallinnan teoriaa. Teorian perusteella valitaan tapauksen käyttöä varten sopivat työkalut ja Kanbania käyttävä projektinhallintaohjelmisto. Tapauksen jokainen merkittävä muutos ja vaihe dokumentoidaan ja kuva kaapataan. Tapauksesta saatavia tuloksia vertaillaan tutkimuksesta saaduilla havainnoilla.

Tämä tutkielma on jäsennetty seitsemään eri lukuun, jotka käsittelevät aiheen teoriaa, työtä ja tuloksia. Luvussa kaksi tarkastellaan projektinhallinnan historiaa ja merkitystä. Tämän jälkeen tarkastellaan perinteisen ja ketterän projektinhallinnan käsitteitä ja eroavaisuuksia. Perinteisten projektinhallintamenetelmien tarkastelua varten tutkielmassa tutkitaan vesiputousmallia ja kriittisen polun menetelmää. Ketterien projektinhallintamenetelmien tarkastelua varten Kanbanin lisäksi tutkitaan Scrumia ja Kanbaniin liittyvää Lean-ajattelua. Scrumin tarkastelu on erityisen tärkeää tutkielman tutkimuksen kannalta, koska Scrum on eniten käytetty projektinhallintamenetelmä ohjelmisto- ja pelinkehityksessä. Tämän lisäksi Scrumin ja Kanbanin muodostama hybridimenetelmä Scrumban on myös yleisesti käytetty menetelmä. Scrum ja Scrumban muodostavat hyvän pohjan Kanbanin toiminnan vertailulle.

Luvussa kolme tarkastellaan ohjelmisto- ja pelinkehityksen käsitteitä. Luvussa tutkitaan kuinka ohjelmisto- ja pelinkehitys eroavat toisistaan ja sitä, voiko molempia käsitteitä käyttää keskenään vaihtokelpoisesti. Tämän lisäksi tarkastellaan mitkä ovat ohjelmisto-

ja pelinalan käytetyimpiä hallintamenetelmiä. Lopuksi perehdytään vielä itsenäisen pelinkehityksen käsitteeseen ja siitä seuraaviin ongelmiin ja etuihin.

Neljännessä luvussa määritellään tutkielman tutkimuksessa käytettävät tutkimusmenetelmät ja aineiston analysointi. Tämän lisäksi tarkastellaan edeltäviä tutkimuksia Kanbanin käytöstä ohjelmistokehityksessä, jolla saadaan hyvä vertailukohde tämän tutkielman tutkimukseen.

Viides luku käsittelee tutkielman tapauksen kulkua. Luvussa esitellään aluksi käytössä olevat ohjelmistot ja peliaihe. Ohjelmistojen esittelyn jälkeen luvussa seurataan tapauksen kulkua ja mahdollisia ilmaantuvia ongelmia sekä menestyksiä.

Luvussa kuusi tarkastellaan tapauksesta saatavia tuloksia. Tulokset on jaettu kahteen osaan: Kanbanin soveltuvuus ohjelmistokehityksen projektinhallintamenetelmänä ja itsenäisesti toteutetussa pelinkehityksessä saatavien tuloksien tarkastelu. Viimeisessä luvussa tarkastellaan tapauksesta saatuja tuloksia ja vertaillaan tutkimuksen tuloksia teorian, käytännön ja jatkotutkimuksien kannalta.

2 Projektinhallinta

Tässä luvussa tarkastellaan projektinhallinnan historiaa, sen tarkoitusta ja kuinka projektinhallintamenetelmät ovat kehittyneet tukemaan ohjelmistokehityksen tarpeita. Luvussa esitellään perinteisten ja ketterien projektinhallintamenetelmien pääasiat, erot ja niiden hyvät sekä huonot puolet ohjelmistokehityksen näkökulmasta. Esimerkkinä perinteisistä projektinhallintamenetelmistä tässä luvussa tarkastellaan vesiputousmallia ja kriittistä polkua. Ketterien projektinhallintamenetelmien esimerkkinä esitellään Scrum, Kanban ja Scrumban. Luvussa tarkastellaan myös Lean-ajattelua, johon työssä käytetty Kanban menetelmä perustuu.

2.1 Projektinhallinnan historia

Projektinhallinnantyökalut ja menetelmät ovat olleet ihmiskunnan käytössä jo useita tuhansia vuosia. Suuret ihmisten tuottamat monumentit, kuten Kiinan muuri ja Gizan pyramidit ovat muinaisten projektien tuottamia tuloksia, joita ihmiset voivat vieläkin käydä katsomassa (Seymour, & Hussein, 2014). Valitettavasti näiden suurten projektien dokumentointi on kadonnut ajan myötä, tai niitä ei ole tehty ollenkaan. Yleisin syy dokumentoinnin puutteesta oli se, että projekteja toteuttavat käsityöläiset saattoivat olla huonosti koulutettuja. Paremman opetuksen saanut ylhäisö ei ollut kiinnostunut projekteissa käytetyistä metodeista. Ylhäisölle tärkeintä oli vain projektien tuottama valmis tuote (Seymour, & Hussein, 2014).

Modernin projektinhallinnan alkuajasta eivät historioitsijat ole päässeet yhteisymmärrykseen. Chiun mukaan moderni projektinhallinta alkoi 1800-luvun loppupuolella. Chiu pitää Henry Ganttia ja Henri Faoylia modernin projektinhallinnan luojina (Chiu, 2010). Haughey pitää Ganttia ja Fayolia tärkeinä projektinhallinnan edistäjinä, mutta hänen mielestään moderni projektinhallinta sai alkunsa vasta 1950-luvulla (Haughey, 2014). Haughey pitää kriittisen polun menetelmää modernin projektinhallinnan alkukohtana, ja

tämän takia tässä tutkielmassa tarkastellaan kriittistä polkua yhtenä perinteisenä projektinhallintamenetelmänä.

Henry Ganttin luoma Gantt-kaavio on yksi moderneista työkaluista, joita käytetään projektienhallintaan. Gantt-kaavion käyttö johti moniin erilaisiin perinteisiin projektinhallintamenetelmien luomiseen. Esimerkiksi Hooverin pato Nevadassa on yksi Gantt-kaaviota käyttäneiden projektien tuottama tuotos, joka on vieläkin toiminnassa. Gantt-kaaviota ja monia muita vastaavia työkaluja käytetään perinteisissä projektienhallintametodeissa, joissa projektin kulku muistuttaa vesiputouksen kaltaista menetelmää. (Seymour, & Hussein, 2014). Vesiputouksen kaltaisessa projektissa projekti aloitetaan täsmällisellä suunnittelulla ja tutkimuksella. Tämän jälkeen projektin vaiheet virtaavat eteenpäin kuin vesiputous, eli vaihe *A* johtaa vaiheeseen *B*, ja niin edelleen. Vesiputouksen tapaan projekti ei virtaa takaisin päin, eli virheiden sattuessa on hankala palata projektin edeltäviin vaiheisiin. Tämä joustamaton malli toimii hyvin projekteissa, jotka ovat yksinkertaisia ja joiden tekemisestä projektien työntekijöillä on jo paljon kokemusta. Vesiputousmallin joustamattomuus tosin aiheuttaa ongelmia kasvavan ohjelmistokehityksen markkinoilla, joten uusien joustavien projektinhallinnan työkalujen kehitys on nähnyt suuren kasvun.

2.2 Projektinhallinnan tehtäviä ja tuotoksia

Projektinhallinta on erilaisten prosessien, metodien, tietojen, taitojen ja kokemuksen soveltamista projektin tavoitteiden saavuttamiseen sovittujen projektin parametrien sisällä. Projektin parametreihin sisältyvät projektin erilaiset resurssit, kuten aika, raha ja työvoima. Projektin parametreihin sisältyvät myös projektin riskit ja laatu (Project Management Institute. 2020). Projektinhallinnan standardoimista varten on amerikkalainen yleishyödyllinen organisaatio, Project Management Institute (PMI), luonut projektinhallinnan viisi vaihetta. Nämä projektinhallinnan viisi vaihetta muodostavat erilaisten projektinhallintametodien selkärangan (Eby, 2018). PMI on myös julkaissut projektinhallinnanoppaan nimeltä: "*A Guide to the Project Management Body of Knowledge"* (PMBOK). Oppaan tarkoitus on luoda standardisoidut perusteet projektinhallinnalle, jota voidaan

käyttää jokaisella toimialalla maailmanlaajuisesti (Project Management Institute, 2017). Projektinhallinnan viisi vaihetta on kuitenkin vain standardointi ehdotus, ja sen takia muitakin projektinhallinnan vaihemenetelmiä on olemassa. Esimerkiksi Amerikkalaisen Lucid Softwaren kirjoittamassa blogissa projektinhallinnassa on käytössä vain neljä vaihetta. Puuttuva vaihe Lucid Softwaren projektinhallinnassa on projektin suorituskyky ja ohjausvaihe, joka on sulautettu osaksi projektin toimeenpano- ja lopetusvaiheita (Lucidchart Content Team, 2020). Projektin suorituskyvyn tarkkailu ja ohjaaminen on siis vain muiden vaiheiden aikana tapahtuva prosessi, joka ei vaadi omaa vaihenimitystä.

Kuvio 1. Projektinhallinnan viisi vaihetta (mukailtu lähteestä Eby, 2018).

Ensimmäinen projektinhallinnan vaihe on projektin käsite ja alullepaneminen. Ensimmäisessä projektinhallinnan vaiheessa määritellään projektin karkeat tavoitteet. Vaiheen tarkoitus on määrittää projektin toteuttamiskelvollisuus, eli kuinka kannattavaa ehdotetun projektin tuottaminen on. Ohjelmistokehityksessä tässä vaiheessa tarkastellaan esimerkiksi järjestelmien ja ohjelmien vaatimuksia.

Toinen projektinhallinnan vaihe tapahtuu, kun ensimmäisen vaiheen toteuttamiskelvollisuus on todettu olevan kannattava. Toisessa vaiheessa määritellään projektin tarkemmat suunnitelmat eli projektin laajuus, aikataulut, kesto, tavoitteet ja riskienhallinta. Projektissa käytössä oleva projektinhallintamenetelmän mukaan vaiheen vaatimukset

voivat vaihdella melko paljon. Esimerkiksi vesiputousmallissa dokumentaatiot ja kaikki projektin tehtävät pitää suunnitella tarkasti etukäteen. Ketterässä Kanban menetelmässä tehtäviä puolestaan voidaan lisätä tarpeen mukaan projektin edetessä.

Kolmas projektinhallinnan vaihe käsittelee projektin tuotteen teko vaihetta. Tässä vaiheessa toteutetaan projektin suunnitteluvaiheessa määritellyt tavoitteet noudattaen sovittuja käytäntöjä. Vaihe sisältää projektin tekovaiheessa toteutetut kokoukset ja suoritusraportoinnit.

Neljäs projektinhallinnan vaihe tapahtuu yhtä aikaa kolmannen projektinhallinnan vaiheen kanssa. Neljännessä vaiheessa tarkastellaan projektin tavoitteiden toteutuksen laatua. Tämä sisältää projektin tavoitteiden toteutuksen tarkastelua, resurssien käytön seurantaa ja mahdollisten projektin muutoksien dokumentaation. Tämä projektin valvontavaihe on tärkeä vaihe ongelmien korjaamiseen ja ehkäisemiseen. Hyvin toteutetun valvonnan avulla voidaan estää projektin epäonnistuminen.

Viides ja viimeinen vaihe on projektin lopetusvaihe. Tässä vaiheessa projektin valmis tuote toimitetaan eteenpäin. Mahdollisten palkattujen urakoitsijoiden viralliset sopimukset päätetään, projektin jäsenten palkitaan ja käydään läpi projektin jälkiselvittely. Projektin jälkiselvittely on tärkeä osa viimeistä vaihetta, koska selvittelyssä esille tulleet muutosehdotukset ja virheet voidaan ottaa käyttöön seuraavaa projektia varten.

Projektinhallinnan yksi yleisimmistä haasteista on määrittää mitä vaaditaan projektilta, jotta sitä voidaan kutsua menestyneeksi. Menestyksen määrittämiseen on hyvä laatia projektille päämäärät, joiden tavoittaminen tarkoittaa onnistunutta projektin tuotosta. Päämäärien tavoitteita voidaan määritellä vastaamalla kolmeen eri kysymykseen:

- *Miltä näyttää menestynyt projekti?*
- *Miten projektin menestyneisyys mitataan?*
- *Mitkä tekijät vaikuttavat projektin menestykseen?*

Vastaukset näihin kysymyksiin pitää dokumentoida hyvin, koska ne muodostavat menestyvän projektin selkärangan (Project Management Institute, 2017). Projektin menestymiseen voidaan lisätä tarkempia kysymyksiä, jolla voidaan mitata esimerkiksi menestyneen projektin taloudelliset kustannukset ja tuotteen laatu.

Projektinhallintamenetelmillä tuotetaan projektin avuksi useita erilaisia työkaluja ja dokumentointeja. Projektia aloitettaessa pitää luoda projektisuunnitelma, joka kattaa koko projektin keston. Valitun projektinhallintamenetelmän mukaan projektisuunnitelman tarkka kattavuus voi vaihdella melko paljon. Esimerkiksi perinteisillä projektinhallintamenetelmillä projektinsuunnitelman on oltava huomattavasti kattavampi, kuin ketterillä projektinhallintamenetelmillä. Projektisuunnitelmaan sisältyy myös riskienhallintasuunnitelma ja resurssienhallintasuunnitelma. Projektin riskien ja saatavilla olevien resurssien kartoittaminen on tärkeä osa menestyvää projektia. Projektisuunnitelmaan sisältyy myös monia erilaisia aikatauluja ja tehtävälistoja.

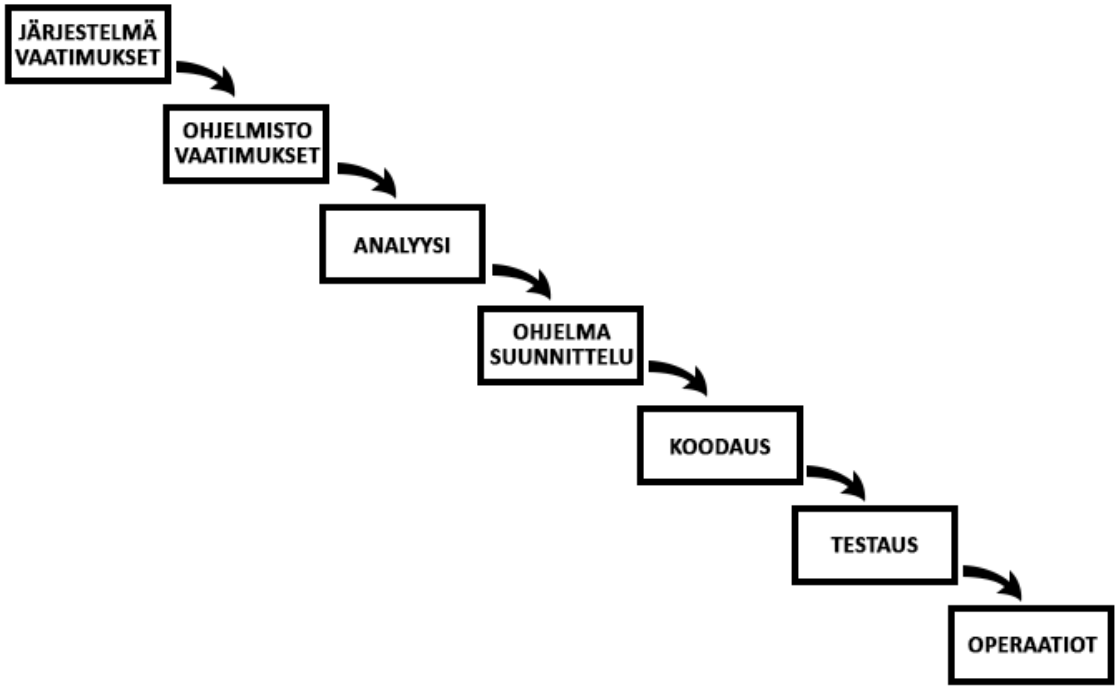
2.3 Perinteinen projektinhallinta

Perinteinen projektinhallinta on maailmanlaajuisesti käytetty projektinhallintatapa, joka seuraa tarkasti projektinhallinnan yleisesti tiedostettuja vaiheita. Perinteistä projektinhallintaa käytetään pääsääntöisesti projekteissa, jonka vaiheet suoritetaan tarkassa järjestyksessä, ja jossa ei odoteta tapahtuvan paljon muutoksia. Perinteisissä projektinhallintamenetelmissä projektin dokumentaatio ja suunnittelu ovat erityisen tärkeäitä perinteisten menetelmien joustamattomuuden vuoksi. Huonosti suunniteltu dokumentaatio saattaa johtaa siihen, että mahdollisen virheen tai arvaamattoman tekijän ilmestyminen saattaa pysäyttää koko projektin. Pahimmassa tapauksessa projektissa joudutaan palaamaan takaisin suunnitteluvaiheeseen, joka tulee maksamaan projektille huomattavan määrään aikaa ja rahaa.

2.3.1 Vesiputousmalli

Vesiputousmalli on yksi tunnetuimmista perinteisistä projektinhallintamenetelmistä. Tämän takia vesiputousmallia saatetaan virheellisesti kutsua perinteisen projektinhallinnan synonyymina (Wrike, 2020). Vesiputousmallin perusidea on projektin vaiheiden seuraaminen tiukassa peräkkäisjärjestyksessä. Vesiputousmallin vaiheet suunnitellaan huolellisesti etukäteen ennen toimeenpanovaihetta. Vesiputousmalli on saanut nimensä vesiputouksen kaltaisesta ulkonäöstä. Vesiputouksen veden virtaamista voidaan verrata projektin vaiheiden etenemiseen. Tämä tarkoittaa käytännössä siis sitä, että vesiputouksen tapaan projekti ei voi ”virrata” takaisinpäin. Tämän takia projektin suunnitteluvaihe on elintärkeä projektin menestymiselle, koska virheiden sattuessa projektissa voi olla hyvin hankalaa palata takaisin suunnitteluvaiheeseen.

Vesiputousmallin kaltaisia menetelmiä on projektinhallinnassa ollut käytössä useita vuosia. Ensimmäinen virallinen kuvaus vesiputousmallin toiminnasta tosin tapahtui vasta 1970-luvulla tohtori Winston W. Roycen kirjoittamassa artikkelissa suurten ohjelmistojärjestelmien kehityksestä. Artikkelissa Royce ei tosin mainitse vesiputous termiä, vaan ensimmäinen maininta vesiputousmallin nimityksestä katsotaan tapahtuneen vuonna 1976 Thomas E. Bellin ja T. A. Thayerin kirjoittamassa artikkelissa: ”*Software requirements: Are they really a problem?”* (Airbrake, 2016). Roycen artikkeli sisältää kuvioita, jossa Royce esittelee erilaisia ohjelmistokehityksen vaiheiden konsepteja. Roycen tuottamat konseptit ovat hyvin samankaltaisia nykyään tunnetun vesiputousmallin kanssa.

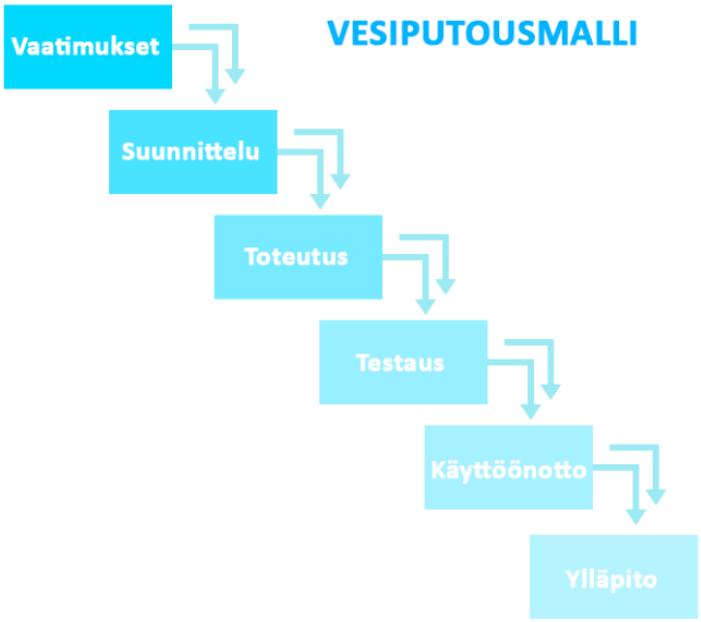


Kuvio 2. Roycen kehittelemä konsepti ohjelmistokehityksen vaiheista (mukailtu lähteestä Royce, 1970).

Vesiputousmallissa on useita vaiheita, jotka suoritetaan peräkkäisjärjestyksessä. Vaiheiden määrä ei tosin ole lukkoon lyöty konsepti. Vaiheiden määrää voi vaihdella vesiputousmallin käyttötarkoituksen mukaan. Esimerkiksi Roycen kehittelemät konseptit vesiputousmallista ohjelmistokehitykseen sisältää vaiheet *system requirements* (järjestelmä vaatimukset) ja *software requirements* (ohjelmisto vaatimukset). Nämä kaksi vaihetta voidaan tiivistää vain yhteen vaatimusten määrittelyvaiheeseen. Muokattuja vesiputousmallin esityksiä on monenlaisia. Monessa mallissa kuitenkin on vähintään viisi päävaihetta: *Vaatimukset*, *suunnittelu, toteutus, testaus* ja *ylläpito.* Kolmannessa kuviossa on esiteltynä nämä viisi päävaihetta ja myös kuudes *käyttöönotto* vaihe.

Vesiputousmallin haittapuolena on mallin jäykkyys, eli mallin peräkkäisjärjestys aiheuttaa ongelmia projektin joustavuuteen. Ongelmien sattuessa useasti joudutaan palaamaan mallin alkuun ja aloittamaan mallin tapahtumasarja uudelleen. Projektin vaatimukset suunnitellaan vesiputousmallissa huolellisesti ennen tuotannon aloittamista. Esimerkiksi kesken projektin toimeksiantajan esille tuomia ehdotuksia ja muutoksia on

hankalaa lisätä kesken kaiken. Projektin laajuus on täten hyvin valmiiksi suunniteltu, ja laajuuden muokkaaminen kesken projektia ei ole helppo toteuttaa. Tämä voi tosin olla myös mallin hyvä puoli, koska mallin jäykkyys estää mahdollisen projektin laajuuden lipsumisen. Tämä ongelma tunnetaan englanninkielisenä *scope creep* terminä. Vesiputousmallin vaiheiden peräkkäisjärjestys voi myös aiheuttaa turhaa hukkaa, koska vaiheet eivät voi tapahtua päällekkäin eli samaan aikaan.



Kuvio 3. Esimerkki vesiputousmallin kuudesta eri vaiheesta.

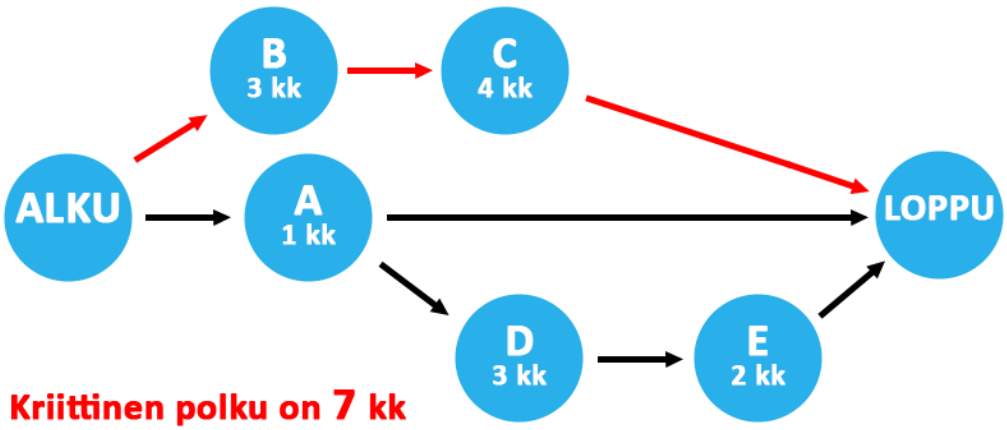
Vesiputousmallin vaatimusten määrittely ja tarkka suunnittelu ovat vesiputousmallin vahvoja puolia. Kattavan dokumentaation avulla projektin jäsenet pystyvät tarkastelemaan kaikkia tulevia tehtäviä etukäteen, ja tämä myös parantaa jäsenien tietoisuutta projektin vaatimuksista. Vaiheiden peräkkäisjärjestys on myös yksinkertaista visualisoida ja ymmärtää, joten vesiputousmallin opettaminen projektin jäsenille on helppoa. Peräkkäisjärjestys myös vahvistaa hyvien työkäytäntöjen oppimista. Esimerkiksi ohjelmistokehityksessä vesiputousmalli vahvistaa koodauksen hyvää suunnittelua ennen koodauksen aloittamista. Vesiputousmalli myös asettaa projektille tarkat aikarajat ja tavoitteiden ajallisen saavuttamisen

2.3.2 Kriittisen polun menetelmä

Critical Path Method (CPM) eli kriittisen polun menetelmä on perinteinen projektinhallintamenetelmä, jossa projektin vaiheet toteutetaan peräkkäisjärjestyksessä. Kriittinen polku on parhaiten tunnettu visuaalisesta kaaviostaan, mutta kriittistä polkua voidaan käyttää ilman polun visualisointiakin. Kriittisen polun kaavan visualisointi on tosin hyvä keino esittää projektin kulku visuaalisesti projektin jäsenille, joten kaavan visuaalinen luominen on kuitenkin suositeltavaa. Kriittisessä polussa määritetään kaikki projektin tavoitteen saavuttamiseen liittyvät välttämättömät työtehtävät. Näiden välttämättömien työtehtävien edellytystehtävät pitävät olla valmiiksi määriteltynä, jotta työtehtävät voidaan lisätä kaavioon. Kriittinen polku on perinteisten projektinhallintamenetelmien tapaan melko jäykkä projektinhallintamenetelmä, joten muutoksia polkuun on hankala tehdä työn aloittamisen jälkeen. Välttämättömien työ- ja edellytystehtävien määrittelemisen jälkeen työtehtäville annetaan jokin visualisointia helpottava symboli, kuten numero tai kirjain. Työtehtävä lisätään tämän jälkeen kaavioon annetulla symbolilla, joka ympyröidään (voi käyttää myös muita muotoja). Ympyröidyn tehtävän viereen tai itse ympyrän sisälle lisätään myös tehtävän odotettu kestoaika (Levy, Thompson, & Wiest, 1963). Tehtävän kestoajalle määritetään myös *slack* eli pelivara, joka on tehtävälle varattu lisäaika mahdollisia viivästyksiä varten (Kukhnavets, 2019). Tehtäville varattu pelivara lasketaan mukaan projektin kestoon. Kriittisessä polussa on aloitusympyrä ja lopetusympyrä, johon lisätyt tehtävät liitetään nuolilla. Nuolen suunta osoittaa peräkkäisjärjestyksen. Esimerkiksi jos nuoli osoittaa tehtävästä *A* tehtävään *B*, tämä tarkoittaa siis sitä, että tehtävää *B* ei voi tehdä ennen *A*:n valmistumista. Tehtävien lisäyksien ja järjestyksien määrittelyn jälkeen kriittisessä polussa lasketaan projektin kesto. Kriittiseksi poluksi kutsutaan pisintä mahdollista polkua projektin alusta loppuun, ja sillä ilmaistaan projektin minimi kestoaika (Levy, Thompson, & Wiest, 1963). Kriittisen polun määrittelyn jälkeen projektinhallinta keskittyy hallitsemaan tätä polkua vesiputousmallin tapaisella peräkkäisjärjestyksellä.

Kriittinen polku sai alkunsa, kun amerikkalainen yritys E. I. du Pont de Nemours and Company (DuPont) päätti kehittää UNIVAC I (**UNIV**ersal **A**utomatic **C**omputer **I**) tietokoneelle

menetelmiä, joilla tietokone pystyisi avustamaan työntekijöitä projektinhallinnassa. Menetelmien tekoon DuPont palkkasi Morgan Walkerin ja pian Walkerin mukaan liittyi Remington Randin työntekijä, James E. Kelley. DuPont ja Remington Rand aloittivat projektin kriittisen polun menetelmän kehitykseen vuonna 1957 Delawaressa järjestetyssä kokouksessa. Projektissa kävi ilmi, että kriitisen polun laskutoimitukset DuPontin rakennusaikataulussa olivat liian isoja UNIVAC I tietokoneelle. UNIVAC I oli liian hidas laskutoimitusten suorittamiselle ja DuPontin aikataulujen laskeminen kesti tietokoneelta useita satoja tunteja. Vuoteen 1959 mennessä sekä DuPont, että Remington Rand olivat jättäneet kriittisen polun kehittämisen taakseen (Weaver, 2006). John W. Mauchly aloitti kriittisen polun jatkokehityksen samana vuonna 1959, ja hänen työnsä avulla kriittisen polun menetelmän käyttö alkoi yleistyä (University of Pennsylvania, 2020).



Tehtävä	Aktiviteetti	Järjestys	Kesto (kk)
A	Tehtävä kuvaus A	Ei edeltäjiä	1
B	Tehtävä kuvaus B	Ei edeltäjiä	3
C	Tehtävä kuvaus C	Aloitetaan kun B on valmis	4
D	Tehtävä kuvaus D	Aloitetaan kun A on valmis	3
E	Tehtävä kuvaus E	Aloitetaan kun D on valmis	2

Kuvio 4. Yksinkertainen esimerkki kriittisen polun menetelmän kaaviosta.

Kriittisen polun menetelmän heikkoutena ovat monimutkaiset projektit, joissa on paljon erilaisia tehtäviä. Kriittisen polun menetelmän historian katsauksessa kävi ilmi, että heikoilla tietokoneilla oli hankaluuksia tehdä kriittisen polun tehtävien laskutoimituksia. Moderneilla tietokoneilla laskutoimitukset onnistuvat sujuvasti, mutta monimutkaisten

projektien hallinta on riippuvainen tietokoneiden ohjauksesta. Suurien projektien monimutkaisuuden takia voi kriittisen polun menetelmän opettaminen uusille projektin jäsenille olla haastavaa. Kriittisen polun menetelmä on perinteisten projektienhallintamenetelmien tapaan melko jäykkä menetelmä, eli menetelmä mukautuu muutoksiin kesken projektin melko huonosti. Ongelmien tai muutoksien sattuessa kesken projektin koko kriittisen polun menetelmän kaavio pitää luoda uudelleen. Tämä saattaa aiheuttaa kasautuvan ongelman. Uuden tehtävän lisääminen kaavioon saattaa muuttaa kriittisen polun pituutta, joka taas vaikuttaa koko projektin kestoon. Projektin keston muuttuessa, saattavat jotkin tehtävät seisahtua, aiheuttaen projektille turhaa ajallista ja rahallista hukkaa (CPM Scheduling, 2019). Ohjelmistokehityksessä saattaa kriittisen polun menetelmä aiheuttaa ongelmia tehtävien keston kanssa. Ohjelmistokehityksessä saattavat ohjelmoinnin vaiheet ja testaus useasti kestää odotettua kauemmin, ja viivästys saattaa olla pidempi kuin tehtävälle suunniteltu pelivara.

Kriittisen polun menetelmän hyvä puoli on projektin vaatimusten huolellinen läpikäynti. Kriittisen polun menetelmä vaatii jokaisen tehtävän listaamisen ja niiden odotetun kestoajan. Tämän takia koko projektin kulku on tiedossa, ja visuaalisen kaavion luodessa projektin seuraaminen on melko yksinkertaista. Kriittisen polun menetelmän tehtäville määritellään pelivara, joten on varautunut mahdollisiin viivästyksiin. Kriittisen polun määrittäminen mahdollistaa turhan ajallisen hukan välttämistä. Projektin suunnitellut tehtävät tapahtuvat määrättyyn aikaan, määrätyn keston mukaan. Tämän avulla projektissa eri tehtävät eivät odota turhaan edeltävän tehtävän valmistumista. Seuraavaan tehtävän tarvittavat resurssit otetaan käyttöön silloin, kun ne on suunniteltu otettavan käyttöön. Yksinkertaisissa projekteissa kriittisen polun menetelmä toimii hyvin ja sitä pystyy käyttämään ilman tietokoneiden apua.

2.4 Ketterä projektinhallinta

Ketterä projektinhallinta keskittyy projektien joustavuuteen, jatkuvaan kehittämiseen ja parhaimman mahdollisen tuotoksen tuottamiseen. Projektien joustavuutta ja jatkuvaa

kehittämistä ohjaa toistuva työnteko, jossa projektin työtehtävät jaetaan pieniin osiin. Tätä työtehtävien jakoa kutsutaan iteratiiviseksi lähestymistavaksi, josta ketterät projektinhallintamenetelmät ovat tunnettuja (Kanbanize, 2020a). Kanban on ketterä projektinhallintamenetelmä, mutta Kanban ei käytä iteratiivista lähestymistapaa. Kanban on siitä huolimatta ketterä projektinhallintamenetelmä, koska Kanban täyttää kaikki muut ketterän projektinhallintamenetelmien tunnusmerkit (Agile Manifesto, 2001). Tässä tutkielmassa tarkastellaan Kanban projektinhallintamenetelmää ja Lean-ajattelua tarkemmin vastaavissa teoriakappaleissa.

Isot projektit, kuten esimerkiksi Hooverin pato, käyttivät useita erilaisia perinteisiä projektinhallintamenetelmiä ja työkaluja. Näiden työkalujen ja menetelmien käyttö vähensi ylimääräisiä kuluja, paransi ajankäyttöä ja teki projektien aikataulutuksesta helpompaa. 1990-luvun alkupuolella ohjelmistokehityksen alalla tuli vastaan nopeasti kasvava ongelma. Teknologian ala oli kehittymässä nopeaan tahtiin ja ohjelmistokehitykseen erikoistuneet yritykset eivät pystyneet pysymään kehityksen perässä. Tämä johti siihen, että useat projektit jouduttiin keskeyttämään ja hylkäämään kokonaan. Loppuun asti viedyt projektit olivat jo vanhentuneet ennen julkaisua, eli projektin tuotteet eivät vastanneet enää uusia teknologian standardeja (Seymour & Hussein, 2014). Oli selvää, että perinteisten projektinhallintamenetelmien jäykkyys, pitkä suunnitteluvaihe ja kattavan dokumentaation tarve ei sopinut ohjelmistokehityksen nopeasti muuttuvaan ympäristöön. Utahissa järjestettiin tämän seurauksena *Snowbird* -nimellä tunnettu kokous, joka käsitteli ohjelmistokehityksen projektinhallintaan liittyviä ongelmia. Samat ohjelmistokehityksen ammattilaiset kokoontuivat Utahissa uudestaan vuosi edellisen kokouksen jälkeen. Tämän kokouksen seurauksena luotiin *”Manifesto for Agile Software Development”,* eli ketterän ohjelmistokehityksen julistus (Highsmith, 2001). Näiden kokouksien seurauksena luotiin myös useita erilaisia ketteriä projektinhallintamenetelmiä, kuten Scrum ja Extreme Programming (XP). Ketterän ohjelmistokehityksen julistus sisältää ketterän ohjelmistokehityksen kaksitoista periaatetta, johon ketterän projektinhallintamenetelmät perustuvat. Ketterien projektinhallintamenetelmien ohjeet ja säännöt voivat kuitenkin erota toisistaan melko paljon. Osa ketteristä menetelmistä voivat toimia vain

projektinhallinnan viitekehyksenä, kun taas osa voi antaa suoria sääntöjä erilaisiin toimintatapoihin.

Ketteriä projektinhallintamenetelmiä käytetään usein ohjelmistokehityksessä ja muissa joustavuutta vaativissa projekteissa. Ketterät projektinhallintamenetelmät keskittyvät ryhmän ja asiakkaan yhteistyöhön sekä joustavaan projektinhallintaan. Tämän takia ketterät projektinhallintamenetelmät sopivat hyvin yrityksille, joilla on tiukat aikarajat tai muuttuvat tavoitteet.

2.4.1 Scrum

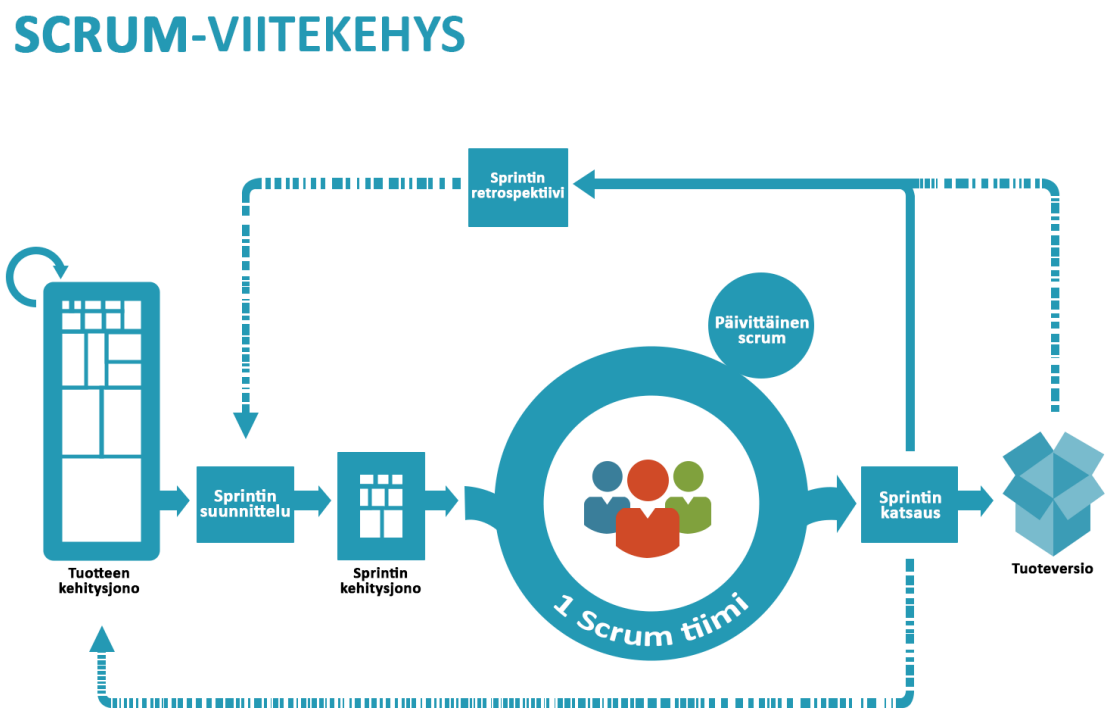
Scrum on yksi parhaiten tunnettu ja eniten käytetty ketterän projektinhallinnan viitekehys. Scrumin suuren suosion takia Scrumia saatetaan virheellisesti kutsua ketterän projektinhallinnan synonyymina (Scrum.org, 2019). Syy tähän sekaannukseen saattaa johtua siitä, että Scrum on keveä ketterän projektinhallinnan sääntökokoelma, eli se muodostaa projektinhallinnan viitekehyksen. Scrum ei ole menetelmäoppi, vaan Scrum on ohjesääntö ketterän projektin hallintaan. Tämä viitekehys rakenne tarkoittaa siis sitä, että Scrumia voidaan helposti yhdistää muiden projektinhallintatapojen kanssa. Projektinhallintamenetelmät, jotka koostuvat eri projektinhallintatavoista kutsutaan hybridi hallintamenetelmiksi. Toinen usein virheellisesti luultu asia on se, että Scrum olisi akronyymi (Chee-Hong, 2018). Nimi Scrum ei ole akronyymi, vaan se on lyhennys termistä *scrummage,* eli aloituskahakka*.* Aloituskahakka termiä käytetään rugby jalkapallolajissa. Akronyymi sekaannus saattaa osittain johtua siitä, että Scrumin luojat Ken Schwaber ja Jeff Sutherland alun perin esittelivät luomaansa ohjelmistokehityksen prosessia suuraakkosilla kirjoitettuna (Verheyen, 2020).

Ken Schwaberin mielestä vesiputousmalli ei soveltunut ohjelmistokehitykseen tarpeeksi hyvin, koska vesiputousmalli on jäykkä peräkkäisjärjestyksellä suoritettava projektinhallinnanmenetelmä. Ohjelmistokehitys oli Schwaberin mielestään liian ennalta arvaamaton, joten projektin työnkulkua oli hyvin hankala ennustaa etukäteen vesiputousmallin

toimintaa varten (Schwaber, 1997). Tämän takia Schwaber lähti kehittelemään menetelmiä, joilla voitaisiin hallita ohjelmistokehityksen epävarmuutta. Hirotaka Takeuchin ja Ikujiro Nonakan kirjoittama artikkeli *"The New New Product Development Game"* (sana *'New'* mainittu otsikossa kaksi kertaa, koska pari halusi painottaa *uutta* sanaa), esitteli termin scrum Schwaberille. Artikkelissa esitellyt asiat muodostivat Scrumin periaatteet. Artikkelissa Takeuchi ja Nonaka kuvailevat uutta lähestymistapaa projektinhallintaan. He kutsuivat tätä rugby jalkapallolajin kaltaiseksi lähestymistavaksi. Tässä lähestymistavassa projektin tiimi yrittää saavuttaa projektin tavoitteet kuten rugbyssä, heittelemällä palloa edestakaisin liikkuen yhdessä eteenpäin (Takeuchi & Nonaka, 1986). Schwaberin mukaan Scrumin kehitykseen liittyi Jeff Sutherland. Yhdessä he julkaisivat vuonna 1995 paperin, jossa he esittelivät Scrumin viitekehyksen (Verheyen, 2020). Schwaberin ja Sutherlandin kehittämän Scrum viitekehyksen käyttö saavutti paljon hyviä tuloksia. Tämän seurauksena Schwaber ja Sutherland päättivät myös kehittää ketterän ohjelmistokehityksen julistuksen.

Scrum oli alun perin kehitetty ohjelmistokehityksen projekteihin, mutta on ajan myötä levinnyt muihinkin aloihin (Schwaber, & Sutherland, 2020). Scrum projektit ovat jaettu pieniin ja ajallisesti lyhyisiin osiin, joita kutsutaan *sprinteiksi.* Sprintit vaativat suunnittelun etukäteen ennen kuin niitä voidaan lähteä toteuttamaan. Sprintin suunnittelussa projektin ryhmä määrittelee mitä työtä ryhmän jäsenet tulevat tekemään sprintin aikana, ja kuinka tämä työ toteutetaan. Näiden kahden kysymyksen vastausta kutsutaan nimellä *sprint backlog,* eli sprintin kehitysjono. Tämän jälkeen sprintti voidaan aloittaa ja sprintissä työskentelevät ryhmät yrittävät suoriutua sprint kertymässä määritellyistä tavoitteista. Sprintteihin sisältyy päivittäisiä kokouksia, joita kutsutaan *scrumeiksi.* Scrum kokousten tarkoitus on käydä projektin vaiheita läpi, ja tarkastella projektin etenemistä. Näiden kokousten avulla pystytään seuraamaan projektin tehtävien edistymistä. Mahdolliset ilmestyneet ongelmat pystytään näin ollen korjaamaan ennen kuin ongelmien vakavuus ehtii kasvamaan. Sprintin valmistumisen jälkeen sprintissä työskennelleet ryhmät esittelevät sprintissä tehdyt tuotokset sessiossa nimeltä *sprint review*, eli sprintin katselmointi. Viimeinen vaihe sprintissä on nimeltään *sprint retrospective,* eli sprintin

retrospektiivi. Sprint retrospektiivissa ryhmät yrittävät tunnistaa sprintin alueita, joita voidaan parantaa seuraavaa sprinttiä varten. (Schwaber, & Sutherland, 2020). Sprint retrospektiivit ovat tärkeitä ketterää projektinhallintaa varten, koska niillä pystytään mukautumaan mahdollisiin muutoksiin ja takaiskuihin.

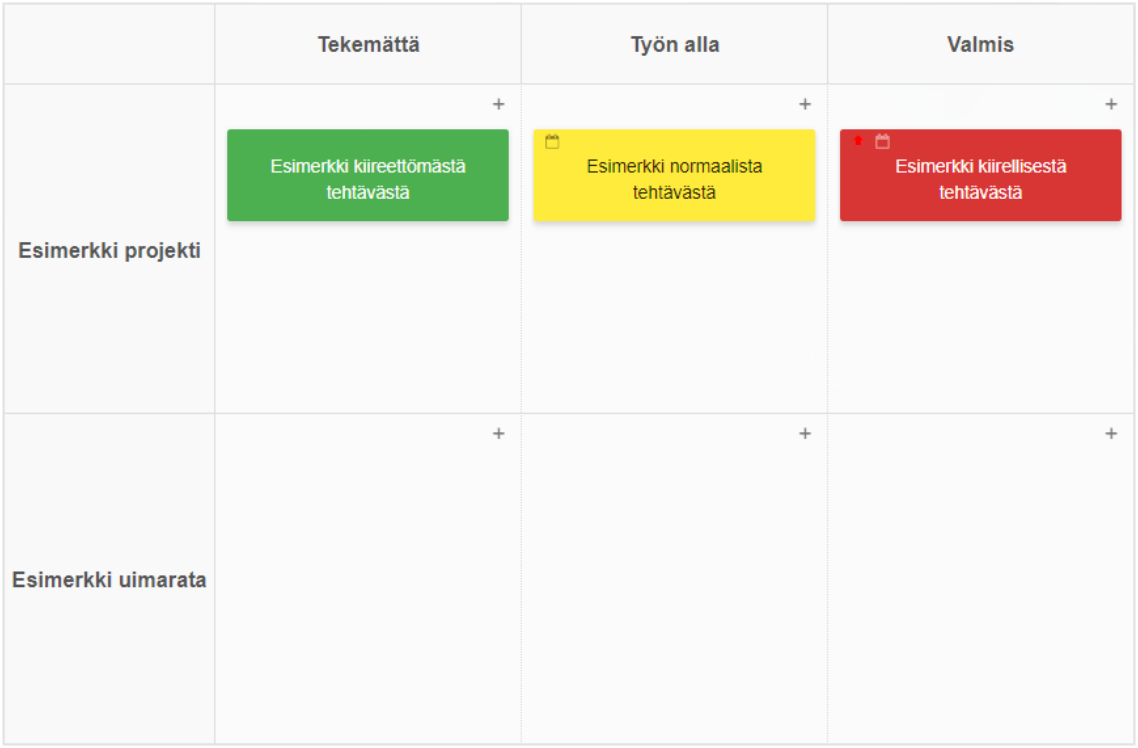


Kuvio 5. Scrum viitekehys (mukailtu lähteestä Scrum.org, 2020)

Scrum viitekehyksessä on kolme roolia; tuotteen omistaja, ryhmä (joka työskentelee projektissa) ja Scrum mestari. Tuotteen omistaja on vain yksi henkilö, mutta tuotteen omistaja edustaa henkilöiden tarpeita, joilla on omistusosuus projektissa, eli usein sidosryhmä. (Schwaber, & Sutherland, 2020). Ryhmärooli sisältää kaikki henkilöt, jotka yrittävät suorittaa sprint kertymät ajallaan. Scrum mestari on vastuussa siitä, että kaikki seuraavat Scrum:n ohjesääntöjä. Scrum mestari itse ei tosin ole osana Scrum ryhmää. Scrum mestarin tehtäviin kuuluvat päivittäisten scrumien järjestäminen, kokousten varaaminen ja osakkeidenomistajille raportointi.

2.4.2 Kanban

Kanban on japaninkielinen sana ja se tarkoittaa kylttiä tai mainostaulua (Dictionary, 2020). Kanbanin näkyvin ja tunnetuin ominaisuus on Kanban-taulu, jota projektin jäsenet täyttävät ohjauskorteilla. Alun perin Kanban-taulu on ollut fyysinen taulu, jota täytettiin muistilapuilla. Tietotekniikan yleistymisen jälkeen taulu on usein korvattu virtuaalisella työkalulla (Rehkopf, 2019). Projektin jäsenet lisäävät Kanban-tauluun ohjauskortteja, jotka sisältävät suunnitellut projektin tehtävät ja tilat.



Kuva 1. Kanban Tool -ohjelmalla tehty Kanban-taulu.

Kanban projektinhallintamenetelmä ja Lean-ajattelu sai alkunsa 1940-luvulla, kun japanilainen autovalmistaja Toyota lähti etsimään ratkaisuja työprosessiensa parantamiseen. Taiichi Ōno oli yksi merkittävimmistä Toyotan työntekijöistä Toyotan työprosessien uudistamisessa. Ōnon avulla Toyota onnistui pelastumaan vararikon partaalta. Taiichi Ōno tutki amerikkalaisten tavaratalojen optimisoituja tuotteen tarjonta prosesseja ja hän mietti, jos kyseisiä prosesseja voisi käyttää projektinhallinnassa (Kanban Tool, 2020).

Tämä johti *Just-in-Time* (JIT*)* eli ”Juuri Ajoissa” -järjestelmään, jossa Kanban toimii tehokkaana ohjaustyökaluna. Suomessa käytetään enemmän termiä JOT eli ”Juuri Oikeaan Tarpeeseen*”* (Logistiikan Maailma, 2020). Juuri ajoissa -järjestelmän tavoitteena on tarjota tuotteita ja raaka-aineita oikean määrän silloin kuin niitä tarvitaan, ja vain sen verran kuin niitä tarvitaan. Ylimääräinen tuotteiden ja raaka-aineiden säilytys ja kuljetus on hukkaa, jota juuri ajoissa-järjestelmä yrittää poistaa. Juuri ajoissa-järjestelmä on hyvin samankaltainen imuohjausmenetelmän (englanniksi *pull control*) kanssa (Logistiikan Maailma, 2020). Imuohjauksessa hankitaan ja käytetään resursseja silloin kuin niitä tarvitaan, kun taas työntöohjauksessa (englanniksi *push control*) ennustetaan tarvittavien resurssien määrä.

Kanban-taulussa on useita erilaisia rakenneosia, jotka yhdessä muodostavat menetelmän jatkuvan työvirran. Kanban-taulun työtehtäviä kuvataan ohjauskorteilla, jotka sisältävät muun muassa työtehtävien tiivistelmät, vastuuhenkilöt ja aikarajat (Rehkopf, 2019). Ohjauskorttien visualisoimat tehtävät vähentävät kokousten tarvetta ja helpottavat työprosessien ymmärtämistä. Digitalisaation vuoksi fyysiset Kanban-taulut ovat siirtyneet ohjelmistojen puolelle, ja tämän vuoksi ohjauskortit ovat saanet useita uusia ominaisuuksia (Kanbanize, 2020b). Esimerkiksi digitaalisissa ohjauskorteissa voi olla kaksi eri puolta. Ohjauskortin etupuolella voi olla esimerkiksi työtehtävien perustiedot ja kääntöpuolella voi olla lisätietoja ja projektin jäsenten kommentteja. Kanban-taulun digitalisointi on myös mahdollistanut reaaliaikaisen tiedonkeruun, jonka avulla ohjelmistot pystyvät avustamaan päätöksien teossa. Kanban-taulu on jaettu eri sarakkeisiin, jotka kuvaavat työnkulkua. Jokaisessa Kanban-taulussa on vähintään kolme eri saraketta; *Tekemättä-, Työn alla-* ja *Valmis* sarakkeet (Rehkopf, 2019). Kanban-tauluun voi lisätä sarakkeita kokemuksen ja tarpeen mukaan. Kanban-taulussa työn alla olevien tehtävien määrää voidaan rajoittaa lisäämällä rajoittimia sarakkeeseen. Kanban-taulussa voidaan myös käyttää uimaratoja (englanniksi *swimlane*), joiden avulla voidaan erottaa taulussa olevat erilaiset työluokat. Esimerkiksi projektissa voi työn alla olla palvelu ja palvelua varten rakennettava laitteisto. Laitteisto ja palvelu ovat erilaisia työluokkia, joten nämä voidaan erottaa taulussa uimaradalla.

Kanban menetelmä keskittyy projektin avainkohtien tekemiseen, joiden avulla voidaan välttää turhaa tehtävien moniajoa. Kanban-taulun ohjauskorttien visuaalisella esityksellä pyritään vähentämään resurssien haaskausta ja parantamaan tehtävien toimitusvirtausta. Näiden ohjauskorttien avulla kaikki projektin jäsenet saavat tarvittavan käsityksen projektin etenemisestä. Tämä myös mahdollistaa reaaliaikaisten ongelmien ehkäisyn. Kanbanin pääero Scrumiin on se, että Kanban keskittyy jatkuvaan työvirtaan (Smartsheet, 2017). Kanbanissa projektin jäsenet voivat muuttaa työtehtävien tärkeysjärjestystä tarpeen mukaan. Scrum keskittyy enemmän toistuvaan työvirtaan, jossa työtehtävät tehdään sprintissä loppuun. Vasta sen jälkeen voidaan korjata ongelmia ja uudelleen järjestää työtehtävien tärkeysjärjestystä.

2.4.3 Lean-ajattelu

Lean-ajattelu on parannusmenetelmä, jonka tavoitteena on auttaa yrityksiä saavuttamaan operatiivisen huippuosaamisen. Operatiivisen huippuosaamisen pääasia on poistaa turhaa hukkaa työprosesseissa ja parantaa työprosessien tehokkuutta. Taiichi Ōnon avulla Toyota kehitti hukantunnistamisjärjestelmän, jossa hukkaa aiheuttavat tekijät ovat kategorisoitu seitsemään eri tekijään (Moujib, 2007). Myöhemmin Lean-ajattelun hukkatekijöihin on lisätty kahdeksas hukan kohde. Kahdeksas hukkatekijä eroaa muista tekijöistä siten, että se ei ole tietty tuotannon prosessi (Skhmot, 2017). Tässä hukkatekijässä työntekijöiden taitoja ei ole esimerkiksi kehitetty antamalla tarvittavaa koulutusta työtehtäviin, tai työntekijöiden taitoja ei ole otettu työssä käyttöön. Työntekijät on voitu myös asettaa työtehtäviin, jotka eivät vastaa työntekijöiden taitojen vahvuuksia.

Projektissa hukkaa voi aiheuttaa turha resurssien kuljetus. Projektin resursseihin voi kuulua esimerkiksi projektin jäsenet, työkalut, varastot ja projektin tuote itse (Moujib, 2007). Tarpeettomat kuljetus- ja liikehukkatekijät liittyvät toisiinsa monella eri tavalla. Tämän takia molempia tekijöitä on hyvä tarkastella yhdessä samalla kerralla. Kuljetus- ja liikehukkaa projektissa voi esimerkiksi aiheuttaa huonosti suunniteltu prosessien fyysinen

sijoitus. Tämän takia on projektia varten tärkeää suunnitella käytössä olevien resurssien sijainti ja resurssien kuljetusmenetelmät. Juuri ajoissa-järjestelmällä pystytään välttämään projektissa turhaa varastoimisen tarvetta ja odottelua. Materiaalin hankinta juuri silloin kun sitä tarvitaan, voi mahdollistaa materiaalin käytön heti ilman turhaa varastointia. Projektissa valmistetun tuotteen ylituotanto on hukkatekijä, joka voi aiheuttaa ylimääräisen varastoimisen tarvetta (Skhmot, 2017). Laatuvirheellisten tuotteiden varastointi voi myös johtaa ylimääräisen varastoimisen tarpeeseen. Ylituotantoa ja laatuvirheiden määrää voidaan vähentää esimerkiksi pienemmillä tuotantoerillä. Pienemmät tuotantoerät mahdollistavat nopeamman reagoimiskyvyn tarvittaviin muutoksiin ja ongelmatapauksiin. Projektin tuotteen ylikäsittely on hukkatekijä, joka voi olla melko hankala havaita ja karsia. Tämän takia on projektin suunnittelussa hyvin tärkeää selvittää asiakasvaatimukset ja projektissa käytetyt prosessit. Monimutkaisten prosessien yksinkertaistaminen on yksi keino välttää tuotteen turhaa ylikäsittelyä.

Kuvio 6. Kahdeksan hukkaa aiheuttavaa tekijää (mukailtu lähteestä Kanban Tool, 2020).

Lean-ajattelu pyrkii poistamaan turhaa hukkaa ja parantamaan työprosesseja viidellä ydin periaatteella:

1. *Määrittele projektin tuotteen tai palvelun arvo kuluttajille.* Arvon määrittely periaatteen tarkoitus on selvittää mitä kuluttajat ovat valmiita maksamaan tuotteesta. Tuotteen arvo voi olla paperilla hyvä, mutta jos kuluttajat eivät ole valmiita maksamaan tuotteelle annettua hintaa, tuotteen arvolla ei ole väliä.

2. *Tunnista tuotteen arvovirta.* Tuotteen valmistaminen vaatii useita eri toimenpiteitä, ja toisen periaatteen avulla yritetään selvittää mitä tuotteen valmistus tulee vaatimaan. Tämä vaihe on tärkeä ylimääräisen hukan poistamiseen ja työprosessien parantamiseen. Projektin hukan kohteita ovat esimerkiksi tuotteeseen liittyvät kuljetuskustannukset, valmistusvirheet ja ylivalmistus.

3. *Luo arvovirta poistamalla hukkaa.* Arvovirran suunnittelun jälkeen on aika laittaa suunnitelmat käytäntöön. Kolmannen periaatteen tarkoitus on luoda jatkuva työvirta poistamalla hukkaa aiheuttavat tekijät.

4. *Anna kuluttajan ohjata virtaa.* Neljännen periaatteen tarkoituksena on antaa kuluttajan ohjata projektin kulkua antaen projektille erilaisia tehtäviä. Tehtävien anto voi tapahtua esimerkiksi Kanban-taulua käyttämällä, johon kuluttaja voi lisätä tehtävälappuja. Tässä vaiheessa täytyy kuitenkin olla tarkkana, ettei anna kuluttajalle liikaa valtaa. Tärkeää on myös välttää tarjoamasta enemmän kuin projektin laajuuteen on suunniteltu.

5. *Jatkuvasti parantele, jotta saavutat täydellisyyden.* Viides ja viimeinen periaate on Lean-ajattelun tunnetuin ominaisuus, täydellisyyden tavoittelu. Lean-ajattelussa on koko ajan tarkoituksena miettiä mikä on projektien tehtävien arvo, ja onko niissä jotain parannettavaa.

Lean-ajattelu on saanut eniten kritiikkiä siitä, miten Lean-ajattelun keskittyminen jatkuvaan paranteluun ja hukan vähentämiseen voi helposti muuttua pakkomielteeksi (Nayab, 2011). Lean-ajattelussa puhutaan useasti täydellisyyden tavoittelusta, mutta todellisuudessa mikään projekti ei voi olla täydellinen. Täydellisyyteen pyrkiminen voi johtaa projektin työntekijöiden kasvavaan stressiin ja keskittymistä projektin vääriin alueisiin. Lean-ajattelu on nimensä perusteella filosofia tai ajattelutapa, jonka takia Lean-ajattelun käyttö projekteissa voi olla hyvin yksilökohtaista.

2.5 Hybridi projektinhallinta

Ketterät projektinhallintamenetelmät sopivat hyvin ohjelmistokehitykseen, mutta joskus projektit vaativat tiukempaa hallitsemista ja ohjeidenmukaisuutta. CAST:n vuonna 2014 tekemän tutkimuksen mukaan ketterän projektinhallintamenetelmän yhdistäminen perinteisen hallintamenetelmän kanssa tuottaa parempilaatuista koodia. Tämä taas johtaa paremmin toimivaan ja turvallisempaan ohjelmistotuotokseen (CAST, 2014). Pääsyy ketterän ja perinteisten menetelmien yhdistämiseen on saada projektiin enemmän ennakoitavuutta perinteisten menetelmien tapaan. Projektiin tarvitaan myös potentiaalia reagoida mahdollisiin muutoksiin ja ongelmiin, joihin ketterät menetelmät soveltuvat hyvin. Hybridimenetelmät ovat myös hyvä tapa opettaa projektin jäsenille erilaisia projektinhallintamenetelmiä ilman, että projektin menetelmiä jouduttaisiin täysin vaihtamaan (Smartsheet, 2020). Project Management Instituten suorittaman yritysten projektinhallintakyselyn mukaan noin 89 prosenttia kaikista kyselyyn vastanneista yrityksistä ovat käyttäneet hybridimenetelmiä projekteissansa (Project Management Institute, 2019). Tämä ei kuitenkaan tarkoita sitä, että kaikki yritykset käyttivät hybridimenetelmiä projektin alusta loppuun. Yritykset ovat voineet käyttää esimerkiksi Scrumia melkein koko projektin keston ajan ja vasta projektin loppuvaiheilla vaihtanut hybridimenetelmän käytäntöihin. Hybridimenetelmien tarjoamat mahdollisuudet oppia uusia projektinhallinta menetelmiä voivat johtaa projektiin sopivamman menetelmän löytämiseen. Tämä johtaa kohennettuun tuotteen laatuun ja resurssien säästämiseen.

2.5.1 Scrumban

Scrumin joustavan viitekehyksen takia Scrumia on yhdistetty moniin eri projektinhallintamenetelmiin. Scrumban ketterä projektinhallintamenetelmä on yksi tunnetuimmista hybridi projektinhallintamenetelmistä, ja se on nimensä mukaan Scrumin ja Kanbanin yhdistelmä (Smartsheet, 2020). Scrumban käyttää Scrumista tuttuja päivittäisiä Scrum kokouksia, joissa suunnitellaan ja tarkastellaan tehtäviä. Toisin kuin Scrumissa, Scrumbanissa ei tehdä projektin tehtäviä sprinteissä, vaan Scrumbanissa tehtävien teko noudattaa Kanban-taulun järjestelmää (Pahuja, 2016). Ketterien projektinhallintamenetelmien tapaan, Scrumbanin ohjeita voidaan tosin soveltaa omaan tarkoitukseen omalla tavalla.

	Scrum	Scrumban
Kertaukset (iteraatiot)	Kyllä (Sprint)	Ei (jatkuva työvirta)
Kehitysjono	Lista arvioiduista ja priorisoiduista tarinoista	Ohjauskortit
Roolit	Tuotteen omistaja, työryhmä, Scrum mestari	Työryhmä ja tarvittavat roolit
Ryhmät	Täytyvät olla ristiinsopivia	Voivat olla erikoistuneita
Ryhmätyöskentely	Yhteistyö tehtävän mukaan	Parveilu (*swarming*) tehtävien saavuttamiseen
Projektin muutokset	Muutokset odottavat seuraavaa sprinttiä	Lisätään Kanban-tauluun tarvittaessa
Tilaisuudet	Päivittäiset scrum-sessiot, sprint suunnittelut, sprint katsaus, sprint retrospektiivi	Päivittäiset scrum-sessiot (suunnittelut, katsaukset ja retrospektiivi tarvittaessa.)
Työnteko	Sprintin johdolla	Työvirran johdolla

Kuvio 7. Scrum ja Scrumbanin eroja.

Scrumban on suunniteltu projekteille, joissa tarvitaan Scrumin rakennetta, mutta myös Kanbanin jatkuva työvirta on projektille tärkeä. Tämän joustavan, mutta jatkuvan työvirran takia Scrumbania käytetään apuna ketterien projektinhallintamenetelmien oppimiseen. Scrumbania voidaan käyttää myös siirtymätyökaluna, jossa projektin tiimi vaihtaa

Scrumista Kanbaniin tai päinvastoin. Scrumbanin käyttö välimenetelmänä on hyvä keino välttää suurimmat häiriöt projektivirrassa. Ketterissä projektinhallintamenetelmissä voidaan käyttää menetelmää, joka tunnetaan nimellä *swarming* eli parveilu. Parveilumenetelmässä projektin jäsenet keskittävät huomionsa tiettyyn tehtävään. Usein menetelmää käytetään tehtävissä, joiden odotetaan aiheuttavan ongelmia tai viivästyksiä. Ryhmänjäsenet kerääntyvät yhteen ratkaisemaan tehtävän mahdollisimman nopeasti. Parveilua voidaan käyttää myös tavallisena käytäntönä monissa eri projektinhallintamenetelmissä, kuten esimerkiksi Scrum ja Kanban. Parveilun hyvänä puolena on sen tarjoama yhteistyö, jonka avulla projektin jäsenet pystyvät tutustumaan toistensa vahvuuksiin ja heikkouksiin. Parveilun huonona puolena on sen hankala toteutus, jos projektin jäsenet eivät ole tottuneet menetelmään (Boiser, 2020). Parveilun käyttö Scrumissa ja Kanbanissa on yleistä, joten menetelmä on myös käytössä Scrumbanissa.

2.6 Projektinhallintamenetelmien hyvät ja huonot puolet

Perinteiset projektinhallintamenetelmät ovat yleisesti melko helppoja oppia ja seurata. Tämän takia ei ole mikään ihme, että perinteiset projektinhallintamenetelmät ovat vieläkin suuressa käytössä maailmanlaajuisesti. Perinteiset projektinhallintamenetelmät tarjoavat projekteille hyvän ja perusteellisen dokumentaation ja suunnittelun. Laadittujen suunnitelmien ja dokumentaatioiden seuraaminen ovat yksi avaintekijöistä perinteisissä projektinhallintamenetelmissä. Perinteisten projektinhallintamenetelmien yksinkertaisuuden takia niiden käyttäminen projektissa helpottaa projektin hallintaa, joka taas johtaa säästettyyn aikaan ja rahaan. Toimialoissa, joissa projektien ei tarvitse olla kovin joustavia perinteiset projektinhallintamenetelmät ovat yleensä turvallisempi ja parempi ratkaisu.

Ketterän projektinhallinnan merkittävin ominaisuus on ketterien menetelmien joustavuus. Joustavuuden avulla projektien jäsenet saavat hallittua työvirtaa paremmin ja mahdolliset ongelmatapaukset projektissa voidaan huomata ja korjata tämän takia nopeammin. Ketterien menetelmien joustavuus myös auttaa tarjoamaan paremman

asiakastyytyväisyyden. Asiakas pystyy olemaan enemmän projektissa mukana, esimerkiksi osallistumalla scrum -kokouksiin. Ketterät menetelmät voivat myös parantaa projektin ryhmien moraalia antamalla projektin jäsenille enemmän vaikuttamismahdollisuuksia. Projektin jäsenien parantunut vaikuttamismahdollisuus voi johtaa jäsenien tuntevan itsensä enemmän hyödylliseksi ja arvostetuiksi.

Ketterien projektinhallintamenetelmien tarjoama joustavuus ei kuitenkaan tule ilman haittapuolia. Ketterien projektinhallintamenetelmien joustavuus on tehty helpottamaan projekteja, joissa tehtävät saattavat muuttua usein, tai joissa saattaa esiintyä paljon arvaamattomia ongelmia. Tämän joustavuuden takia ketterät projektinhallintamenetelmät saattavat aiheuttaa projektin jäsenille turhautuneen olon, kun projektissa saattavat tehtävät pyöriä paikoillaan. Ketterien projektinhallintamenetelmien sääntöjen huonosti noudattaminen saattaa myös johtaa huonoihin tapoihin ja päätöksiin, jotka lisäävät projektin jäsenten turhautuneisuuden tunnetta (Fridman, 2016). Perinteisten projektinhallintamenetelmien suurin etu on niiden kattava dokumentointi ja suunnittelu, jotka helpottavat ehkäisemään väärinymmärryksiä. Ketterät projektinhallintamenetelmät ehdottavat myös tekemään kattavan dokumentoinnin ja suunnittelun, mutta niiden tarve ei ole yhtä kriittinen kuin perinteisissä projektinhallintamenetelmissä. Ketterällä projektinhallinnalla johdettujen projektien vähäisempi dokumentointi ja suunnittelu saattavat johtaa projektin tavoitteen harhautumiseen. Tämä voi johtaa projektin tavoitteen laajuuden kasvamiseen. Kasvaneen tavoitteen laajuuden takia voi projektin aikaraja viivästyä ja projektiin käytettävien resurssien tarve kasvaa suuremmaksi kuin alun perin oli suunniteltu.

Ketterät projektinhallintamenetelmät ovat mahdollistaneet ohjelmistokehityksen alalla palveluiden kehityksen yleistymisen. Ketterien menetelmien avulla voivat yritykset jatkaa projektin tuottaman tuotteen kehitystä palveluna projektin valmistumisen jälkeen. Monet yritykset ohjelmistokehityksen alalla ovat huomanneet, että projektien tuottamat ohjelmat eivät tarvitse olla täysin valmiita projektien loputtua (Hirschtick, 2020). Tietotekniikanala on nopeasti kehittyvä ala, jossa kehitetyt tekniikat saattavat olla

vanhentuneita jo seuraavana vuotena. Tämän takia monet yritykset ovat laskeneet, että on taloudellisempaa julkaista tarpeeksi hyvin toimiva ohjelmisto markkinoille, kuin yrittää kehittää ohjelmistoon kaikki tarpeelliset toiminnot mukaan. Nämä toimivat, mutta ei vielä valmiit ohjelmat, paikataan palvelumallilla myöhemmin asiakaspalautteen ja lisäkehityksen avulla. Yleistynyt palvelukäytäntö on saanut paljon kritisismiä varsinkin pelialalla. Bethesda Game Studios -pelistudion johtava tuottaja Todd Howard vuonna 2019 tehdyssä haastattelussa kertoi heidän julkaisemansa pelin Fallout 76 julkaisusta ja pelin tulevaisuudesta. Haastattelussa Howard antoi kuuluisan lainauksen ohjelmistokehityksen muuttuneesta palvelumallista: "*It's not how you launch, it's what it becomes—".* Howardin mielestä tuotteiden julkaisutilalla ei ole paljon väliä, koska palvelumallin avulla he pystyvät jatkamaan pelinkehitystä useita vuosia julkaisun jälkeen (Tassi, 2019). Bethesdan julkaisema Fallout 76 -peli sai paljon kritisismiä kriitikoilta ja kuluttajilta, koska peli julkaistiin hyvin keskeneräisessä tilassa.

Perinteisten projektinhallintamenetelmien huolellisesti tehty kattava dokumentaatio ja suunnittelu ovat yleisesti helposti ymmärrettäviä ja seurattavia. Tehdyt suunnitelmat yrittävät kattaa kaikki projektin tehtävät ja mahdolliset ongelmat. Kaikkea ei aina välttämättä pysty ennakoimaan ja näiden arvaamattomien ongelmien ratkaiseminen perinteisten projektinhallintamenetelmien jäykässä rakenteessa voi olla ongelmallista. Joissakin ongelmatapauksissa voi projektissa olla pakko hyväksyä mahdollinen vika, koska sen korjaaminen saattaa johtaa koko projektin pysäyttämiseen. Pysäytys voi tulla maksamaan enemmän kuin mitä vian korjaamisesta voisi hyötyä. Perinteisten projektinhallintamenetelmien tuottama tuote lähestyy valmistumista vasta projektin loppupuolella, joten tuotteen toiminnan tutkiminen on hankalaa ennustaa. Perinteinen projektinhallintamenetelmä ei ole hyvä projektinhallintaratkaisu projekteille, joiden tarkoitus on tuottaa palvelu. Projektin ryhmät tulevat ylläpitämään valmista tuotetta projektin jälkeen. Vaikka projektin dokumentointi ja suunnittelu olisi mahdollisimman hyvin tuotettu, on mahdotonta ennustaa minkälainen projektin tuottama palvelu tulee olemaan usean vuoden päästä palvelun julkaisemisen jälkeen.

3 Projektinhallintamenetelmät pelinkehityksessä

Tässä luvussa perehdytään projektinhallintamenetelmien toimintaa pelinkehityksessä ja sitä, kuinka pelinkehitys eroaa ohjelmistokehityksestä. Luvussa myös selvitetään mitkä ovat tällä hetkellä käytetyimmät projektinhallintamenetelmät pelinkehityksessä. Näiden lisäksi luvussa tarkastellaan myös itsenäistä pelinkehitystä.

3.1 Ohjelmisto- ja pelinkehityksen eroavaisuudet

Pelinkehitys on osa ohjelmistokehitystä ja molemmat kehitysalueet jakavat samankaltaiset kehitysprosessit (Bethke, 2003). Nämä kehitysprosessit voivat kuitenkin erota tosistaan melko paljon, jonka takia kaikkia ohjelmistokehityksen prosesseja ei voi välttämättä käyttää tehokkaasti pelinkehityksessä (Murphy-Hill ja muut., 2014; Pascarella ja muut., 2018). Murphy-Hill ja muut (2014) suorittamassa tutkimuksessa saatiin selville, että pelinkehityksen vaatimukset voidaan tiivistää yhteen avainkäsitteeseen; pelien on oltava hauskoja. Ohjelmistokehityksessä keskeisenä käsityksenä ohjelmille ovat tehokkuus, toimivuus ja helppokäyttöisyys. Pelinkehityksen vaatimukset ovat huomattavasti subjektiivisempia, koska hankalakäyttöinen videopeli voi käyttäjän mielestä silti olla hauska. Videopelin hauskuuden määrittely on hankala ongelma, koska siihen ei ole suoraa kaaviota. Videopelien jatko-osissa ongelmana on uusien pelimekaniikkojen kehitys, koska suora kopio edellisestä pelistä voi menettää pelin hauskuustekijän. Samojen mekaniikkojen kopiointi voi aiheuttaa pelaajille yksitoikkoisuuden tunnetta, mutta liialliset pelimuutokset saattavat aiheuttaa alkuperäisen pelin viehätyksen katoamisen (Caruso, 2018). Pelinkehityksen subjektiivisuus on tämän takia yksi suurimmista tekijöistä kehitysprosessien eroavaisuuksiin.

Pelinkehityksessä subjektiivisuuden vaikutus näkyy myös pelitestauksessa. Pelinkehityksessä voidaan testata pelin toimivuutta ohjelmistokehityksen tapaan erilaisilla testausohjelmilla. Testausohjelmien heikkoutena on kuitenkin niiden kykenemättömyys testata pelien hauskuus tekijää (Murphy-Hill, E., Zimmermann, T., ja Nagappan, N., 2014). Pelien

subjektiivisuuden takia pelinkehityksessä voidaan käyttää useita pelitestaajia. Pelitestaajien antaman palautteen avulla voivat pelinkehittäjät tarkastella pelin yhteisiä hauskuus- ja ongelmatekijöitä.

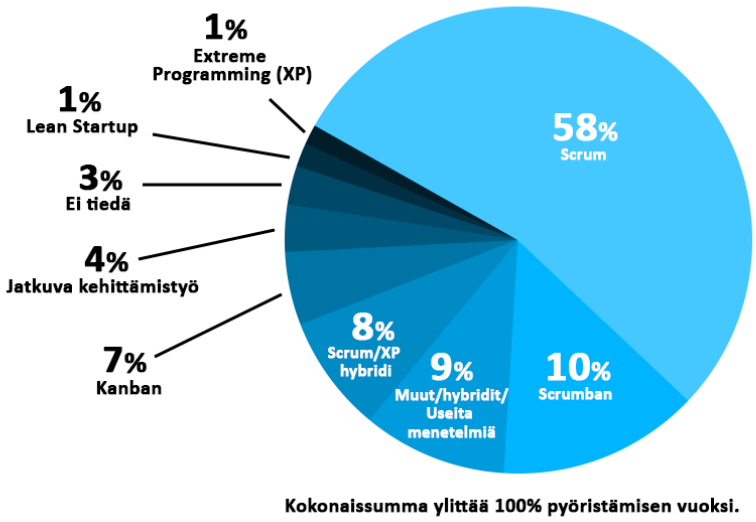
Projektin suunnittelu ja dokumentointi on tärkeää ohjelmistokehityksessä riippumatta siitä, mitä projektinhallintamenetelmää projektissa käytetään. Projektissa tuotettavan ohjelman vaatimukset kartoitetaan ja suoritetaan erilaisia pohjatutkimuksia ohjelman tärkeimmistä ominaisuuksista. Murphy-Hill ja muut (2014) tutkimuksen haastatteluihin osallistuneiden osapuolien mukaan pelinkehityksessä tarkkojen suunnitelmien laatiminen on haaskattua aikaa, koska suunnitelmien tuotokset eivät välttämättä tuota hauskaa peliä. Esimerkiksi videopeliyrityksen Hello Gamesin tuottamassa No Man’s Sky -pelissä planeetoille oli suunniteltu pyörähdysaika, eli planeettojen pyörimien oman akselinsa ympäri. Pelaajatestauksen jälkeen kävi kuitenkin ilmi, että planeettojen pyörähdysaika aiheutti pelaajille navigointiongelmia (Hello Games, 2016). Murphy-Hill ja muut (2014) haastatteluissa kävi myös ilmi, että suunnitelmien ja dokumentaation puute saattoi johtaa pelin tekniseen velkaan.

Ohjelmistokehityksen monipuolisuuden ja laajan yleistymisen takia avuksi on tehty useita erilaisia työkaluja. Nämä työkalut voivat olla esimerkiksi erilaisten yrityksien tarjoamia palveluja tai ohjelmistokehityksen yhteisön jakamia avoimen lähdekoodin ohjelmia. Ohjelmistokehityksen avoimen lähdekoodin työkalujen runsauden ja monipuolisuuden takia pystyvät yritykset ja harrastajat testaamaan erilaisten työkalujen toimintoja ilman rahallista sitoutumista. Tässä tutkielmassa käytettiin avoimen lähdekoodin ohjelmia kuten myös maksullisia palveluja. Pelinkehityksen avuksi on myös luotu useita erilaisia työkaluja. Murphy-Hill ja muut (2014) haastattelun mukaan saattoivat kuitenkin useat eri yritykset kehittää omat yrityksen sisäiset ohjelmat, joita yritykset eivät jaa julkisesti. Pelinkehityksessä pelimoottorit nopeuttavat pelien kehitystä tarjoamalla valmiin ohjelmistokehyksen peleille (Ward, 2008). Pelimoottorit voivat olla yrityksien omia sisäisiä ohjelmistokehyksiä, tai ne voivat olla yrityksien tarjoamia tuotteita (Khan, 2019). Pelimoottorit, kuten esimerkiksi Epic Gamesin tarjoama Unreal Engine ovat ilmaisia ladata

ja käyttää. Epic Gamesille pitää kuitenkin maksaa pelimoottorin käytöstä rojalteja, jos pelimoottoria käyttävän tuotteen bruttotulo ylittää miljoona dollaria (Epic Games, 2021). Yrityksen sisäisesti kehitettyjä pelimoottoreita varten yritykset usein myös kehittävät omia työkaluja tiedostojen kääntämistä ja muokkaamista varten.

3.2 Pelinkehityksessä käytetyimmät projektinhallintamenetelmät

Ohjelmistokehityksen avuksi on luotu useita erilaisia ketteriä projektinhallintamenetelmiä, kuten Scrum ja Extreme Programming. Yhdysvaltalainen yritys Digital.ai julkaisee vuosittain *State of Agile* raportin ketterien menetelmien käytöstä ohjelmistokehityksessä. Raportissa on maailmanlaajuisen kyselyiden tuloksia ketterien menetelmien käytöstä yrityksissä. Vuonna 2020 julkaistussa raportissa 95 % vastanneista yrityksistä ovat käyttäneet ketteriä menetelmiä (Digital.ai, 2020). Samassa raportissa käytetyimpänä ketteränä menetelmänä ensimmäisellä sijalla on Scrum (58 %). Scrum on ollut raporttien tuloksissa ylivoimaisesti eniten käytetty ketterä menetelmä jo useita vuosia. Scrumia seurasi useat erilaiset hybridimenetelmät kuten esimerkiksi Scrumban (10 %). Scrumin ja hybridimenetelmien jälkeen yrityksissä käytetyin ketterä menetelmä on Kanban (7 %).



Kuvio 8. Käytetyimmät ketterät projektinhallintamenetelmät (mukailtu lähteestä Digital.ai, 2020).

Pelin- ja ohjelmistokehityksen eroavaisuuden takia pelinkehityksessä voisi myös olla eroa käytetyimmissä projektinhallintamenetelmissä. Erilaisten pelinkehityskyselyiden perusteella menetelmien valinta ei kuitenkaan eroa paljoa ohjelmistokehityksen menetelmistä (Koutonen, Leppänen, 2013; Schweda ja muut, 2010). Scrum on pelinkehityksessäkin eniten käytetty ketterä menetelmä. Scrumin jälkeen menetelmänä yleisesti seurasi Scrumin sisältävä hybridimenetelmä kuten esimerkiksi Scrumban. Tämä samankaltainen menetelmien valinta johtuu siitä, että ketterät menetelmät pystyvät mukautumaan pelinkehityksen vaatimuksiin luontevasti (Kasurinen, Palacin-Silva, Vanhala, 2017). Ohjelmistokehityksen laajan tutkimuksen takia ketteristä menetelmistä löytyy runsaasti tietoa, jota voidaan soveltaa pelinkehityksen tarpeisiin.

3.3 Itsenäinen pelinkehitys

Tietotekniikan nopea kehittyminen on mahdollistanut tietokonepelien julkaisemisen digitaalisissa latauspalveluissa. Pelinkehittäjien ei tarvitse enää käyttää pelinkehitykseen sijoitettua rahaa fyysisten kopioiden luomista varten. Tämä on helpottanut pienten pelistudioiden pelinkehitystä. Pelistudiot voivat käyttää digitaalisista kopioista säästettyä rahaa esimerkiksi pelin mainostamiseen. Pelinkehitystä on myös nopeuttanut useiden erilaisten pelimoottorien saatavuus. Pelialalle pääsyn kynnystä on varsinkin alentanut ilmaiset pelimoottorit, kuten esimerkiksi Unity ja Unreal Engine (Poole, 2020). Pelimoottorit antavat pelinkehittäjälle käyttöön useita erilaisia työkaluja ja valmiita pohjia, jotka nopeuttavat pelinkehitystä. Esimerkiksi Unity Technologiesin tarjoamassa Unity Asset Storessa voivat käyttäjät jakaa ja myydä valmiiksi tehtyjä pakkauksia. Nämä pakkaukset voivat sisältää esimerkiksi pelinkehitykseen kuuluvia ääniraitoja, tekstuureja tai 3D-malleja. Pakkaukset ovat mahdollistaneet pelinkehityksen, jossa pelikehittäjien ei tarvitse itse tehdä kaikkia pelinkehitykseen liittyviä osia.

Ilmaisten pelinkehityksen työkalujen määrä on helpottanut aloittelijoiden pääsyä pelinkehitykseen mukaan. Itsenäisesti kehitetyt pelit voivat työkalujen avulla muistuttaa

isojen pelistudioiden tuotoksia. Esimerkiksi hyvin tunnettu Minecraft -videopeli oli alun perin vain yhden henkilön kehittämä tuote (Ramée, 2018).

Pelien itsenäinen kehitys alkaa usein harrastuksena tai intohimoprojektina. Tämän takia itsenäisessä pelinkehityksessä ei välttämättä ole käytössä projektinhallintamenetelmiä. Pelien itsenäisessä kehityksessä projektinhallintamenetelmien arvo ei saata ilmentyä pelinkehityksessä ennen kuin sen puute on voinut johtaa teknisen velkaan tai projektin laajuuden lipsumiseen (Estevez, 2015; Bethencourt, 2016). Itsenäisen pelikehityksen ongelmana voi myös olla motivaation puute, joka voi johtua monesta eri yksittäisistä tai yhtenäisistä tekijöistä. Projektin laajuuden lipsuminen voi johtaa turhautuneisuuteen, koska työn määrä kasvaa työnteosta huolimatta. Aikataulujen ja työtehtävien luominen ennalta ehkäisee laajuuden lipsumista. Peliprojektin edistyminen tehtyjen tehtävien visualisoinnilla saattaa innostaa kehittäjää jatkamaan työntekoa, koska hänen työpanoksensa on visuaalisesti nähtävillä. Motivaation puutteeseen voi myös vaikuttaa työkaverien puute ja siitä johtuvan palautteen sekä vuorovaikutuksen menetys.

3.4 Kanban ohjelmistokehityksessä

Kanbanista tehtyä tutkimusta ohjelmistokehityksessä on hyvin niukasti esimerkiksi Scrumiin verrattuna. Ahmadin, Markkulan ja Oivon (2013) tuottamassa kirjallisuuskatsauksessa Kanbanin käytöstä ohjelmistokehityksessä ilmeni, että Kanbanista ohjelmistokehityksessä ei ollut tehty yhtäkään tutkimusta vuodesta 2000 vuoteen 2007. Katsauksessa ilmeni myös, että suurin osa Kanbanista tehdystä tutkimuksesta ohjelmistokehityksessä oli julkaistu vuonna 2011. Yksi näistä tutkimuksista oli Ikonen ja muut (2011) tuottama empiirinen tutkimus Kanbanin käytöstä ohjelmistokehityksen projektinhallinnassa. Ikonen ja muut (2011) tutkimuksen tuloksissa ilmeni, että Kanban toimii erittäin hyvin ohjelmistokehityksen projektin ajantasaisessa visualisoinnissa. Kanbanin heikkoutena kuitenkin ilmeni Kanbanin liiallinen yksinkertaisuus, joten Kanbanin käyttö toimii parhaiten liitettäessä osaksi muita menetelmiä. Scrumban on yksi projektinhallintamenetelmä, joka käyttää hyväksi Kanbanin tarjoamaa projektin visualisointia ja jatkuvaa työvirtaa.

Kanbanin tarjoaman projektinhallinnan visualisoimisen avulla projektissa mahdollisesti esiintyvät hukan lähteet voidaan ehkäistä ennen kuin niistä aiheutuu projektin onnistumista heikentävää haittaa. Ikonen ja muut (2010) tuottamassa tutkimuksessa huomattiin, että visualisuudesta huolimatta projektiin saattoi ilmaantua hukan lähteitä. Hukasta huolimatta Ikonen ja muut (2010) tuottaman tutkimuksen projektin tavoite saavutettiin onnistuneesti. Onnistunut projekti ei täten tarkoita, että projektissa ei olisi ollut ollenkaan hukan lähteitä esillä. Hukan tunnistaminen ja karsiminen on tärkeää onnistuneen projektin työvirralle, mutta kaikkia hukkaa aiheuttavia lähteitä ei projektissa pysty realistisesti poistamaan. Kanbanin jatkuvan työvirran avulla voidaan vähentää viivästyksistä johtuvaa hukkaa siirtämällä työntekijät toiseen työtehtävään, kunnes viivästyksen aiheuttanut ongelma on saatu korjattua.

Kanban-taulussa on mahdollista asettaa projektille tehtävärajoituksen, jossa määritellään kuinka monta työtehtävää voi projektissa olla tekeillä yhtä aikaa. Rajoituksen tarkoituksena on keskittyä projektin tärkeisiin työtehtäviin ja ehkäistä liiallisesta yhtäaikaisesta työnteosta johtuvaa sekasortoa. Sjøbergin (2018) tuottamassa tutkimuksessa Kanbanin työtehtävien rajoittaminen paransi projektin läpimenoaikaa, mutta samalla projektin tuotantokyky heikentyi. Projektin työtehtävien liiallinen rajoittaminen voi jättää osan projektin työntekijöistä ilman työtehtävää. Projektille sopiva työraja muuttuu työntekijöiden ja projektin koon mukaan. Tämän takia työtehtävien rajoittamista varten ei ole yhtä oikeaa vastausta. Projektille sopivan työrajan kehittäminen vaatii tasapainottelua läpimenoajan ja tuotantokyvyn heikkenemisen kanssa.

Granulon ja Tanovićin (2019) tuottamassa tutkimuksessa vertailtiin Kanbanin ja Scrumin toimivuutta ohjelmistokehityksessä. Tutkimuksessa vertailuun käytetyt tulokset saatiin tuottamalla projekti, jossa kehityksen aiheena oli oppimisen hallintajärjestelmän luominen. Projektista saatujen tuloksien mukaan Kanban toimii Scrumia paremmin projekteissa, joissa nopea projektin yksittäisten osien tuotto on tärkeää. Kanban toimii myös Scrumia paremmin, jos projekti ei ole ajallisesti rajoitettu. Kanbanin heikkoutena

tuloksissa huomattiin, että Kanbanin avulla projektin kesto oli hankala arvioida. Scrumin vahvuutena on tarkka projektin aikataulutus, joka mahdollistaa projektin ajallisen valmistumisen.

Schwedan, Winklerin, Biifflin ja Musilin (2010) tuottaman tutkimuksen kyselyssä pyrittiin selvittämään mitkä ovat eniten käytettyjä projektinhallintamenetelmiä pelinkehityksessä. Kyselyn tuloksissa selvisi, että pelinkehityksessä eniten käytetyt projektinhallintamenetelmät mukailevat ohjelmistokehitystä. Ohjelmisto- ja pelinkehityksessä Scrum oli huomattavasti eniten käytetty projektinhallinnantapa, jonka takia Scrumista oli tehty huomattavasti enemmän tutkimuksia Kanbaniin verrattuna.

4 Tutkimusmenetelmä

Tässä luvussa tarkastellaan tutkielmassa käytettyä tutkimusmenetelmää ja käydään läpi edeltävää tutkimusta tutkielman aiheesta. Luvussa myös esitellään tutkimuksessa käytetty laadullinen analyysimenetelmä fenomenologinen analyysi. Lopuksi esitellään fenomenologiseen analyysiin pohjautuva tutkimusprosessi tapaustutkimuksen suorittamiselle.

4.1 Tapaustutkimus

Tämän tutkielman tutkimustapana käytettiin tapaustutkimusta. Tapaustutkimus on tutkimusstrategia, jossa tutkitaan jotain tutkimuksen kohdetta eli tapausta syvällisesti (Jyväskylän yliopisto, 2015a). Tapauksia voidaan kutsua myös englanninkielisellä nimityksellä *case* (Hirsjärvi, Remes ja Sajavaara, 2009).

Tutkimuksen tapauksena oli tietokonepelin kehitys Kanban -projektinhallintamenetelmää käyttäen. Pelinkehitysprojektin edistymistä dokumentointiin ottamalla projektin vaiheista kuvankaappauksia ja kirjoittamalla tärkeimmät havainnot muistikirjaan. Projektin valmistuttua kootut havainnot muodostivat tuloskokonaisuuden, jota voitiin käyttää jo olemassa olevien projektinhallintatutkimuksien vertailuun.

4.2 Aineiston analysointi

Tutkielman tavoitteena oli tarkastella Kanbanin soveltuvuutta ketteränä projektinhallintamenetelmänä ohjelmistokehityksessä. Kanbanin soveltuvuutta varten tutkielmassa suoritettiin tapaustutkimus, jossa Kanbania käytettiin itsenäisessä pelinkehitysprojektissa. Projektista saatuja havaintoja varten tutkielman aineiston analyysimenetelmäksi valittiin fenomenologinen analyysi. Fenomenologisessa analyysissä tutkimuksesta tehdään havaintoja ja pohditaan sekä reflektoidaan tutkijan tutkimuskohteesta saatuja

kokemuksia (Jyväskylän yliopisto, 2015b). Tutkijan täytyy ennen tutkimuksen analyysin tekemistä tuoda ilmi lähtökohtansa tutkimuksen aiheen tutkimisesta. Analysointia varten tutkijan on tiedostettava mahdolliset ennakko-oletukset, jotta analysointiin vaikuttavat vääristymät voidaan ottaa huomioon tuloksien analysoinnissa.

Tutkielman tapaustutkimuksen havainnot kirjattiin muistikirjaan ja tärkeimmistä havainnoista otettiin kuvankaappauksia tutkimuksen jälkeistä analysointia varten. Tutkimuksesta saatuja havaintoja verrattiin jo olemassa oleviin tutkimuksiin Kanbanin käytöstä ohjelmistokehityksessä. Tutkimuksen tuloksia verrattiin myös olemassa oleviin tuloksiin itsenäisestä pelinkehityksestä, jotta Kanbanin soveltuvuutta voidaan tarkastella ryhmäkoon mukaisesti.

4.3 Tutkimusprosessi

Tutkielman tutkimusprosessi aloitettiin aiheen valinnalla ja tutkielmalle annetun käytettävän ajan määrittelyllä. Käytettävän ajan perusteella voitiin tutkielmalle luoda suuntaa antava aikataulu tutkimuksen eri vaiheille. Aikataulutuksen luonnin jälkeen vuorossa oli aiheen aineistoon perehtyminen, jotta voitiin määritellä tutkimuksen tutkimuskysymykset ja tutkimuksen tarve. Alustavan aineistokatsauksen jälkeen luotiin teoriapohja, joka toimi aineiston tutkimisen ja analysoinnin tukena.

Tutkimuksessa käytettyjen ohjelmistojen ja palveluiden valinta perustui jo aikaisempaan kokemukseen ohjelmoinnista, sekä teoriakatsauksessa saadusta käsityksestä tutkimuksen aiheesta. Kanban-taulun tarjoava Kanban Tool otettiin käyttöön, jotta tapauksen suunnittelun aikana voidaan lisätä suunnitellut tehtävät suoraan Kanban-tauluun. Tapauksen suunnittelussa pyrittiin määrittelemään etukäteen mahdollisimman monta työtehtävää, jotta tauluun saataisiin projektille suotuisa lähtökohta. Tehtävien suunnittelua varten ei kuitenkaan käytetty paljon aikaa, koska tutkimuksessa oli tärkeää tarkastella Kanbanin kykyä mukautua odottamattomiin muutoksiin projektissa.

Kuvio 9. Tutkielman tutkimusprosessin kulku.

Tutkimuksen tapauksen kulkua dokumentoitiin tarkasti ja jokaisesta muutoksesta tai tärkeästä vaiheesta otettiin kuvankaappauksia ja muistiinpanoja. Tapauksen valmistuttua kerättiin dokumentaatiossa ilmaantuneet tärkeimmät asiat yhteen ja verrattiin niitä jo olemassa oleviin aineistoihin. Vertauksessa ilmaantuneet avainasiat muodostivat pohjan tulosten analysointiin sekä niiden merkityksen teorialle, käytännölle ja jatkotutkimuksille.

5 Toteutus

Tässä luvussa tarkastellaan tutkielmassa käytössä olleita ohjelmistoja ja käsitellään tutkielman projektin toteutusta. Projektissa käytettyjen ohjelmien valinnat perustellaan tutkielmassa käytössä olevaa fenomenologista analyysia varten. Projektista saadut tärkeimmät havainnot ovat esitetty visuaalisesti kuvankaappauksilla.

5.1 Projektissa käytetyt ohjelmistot

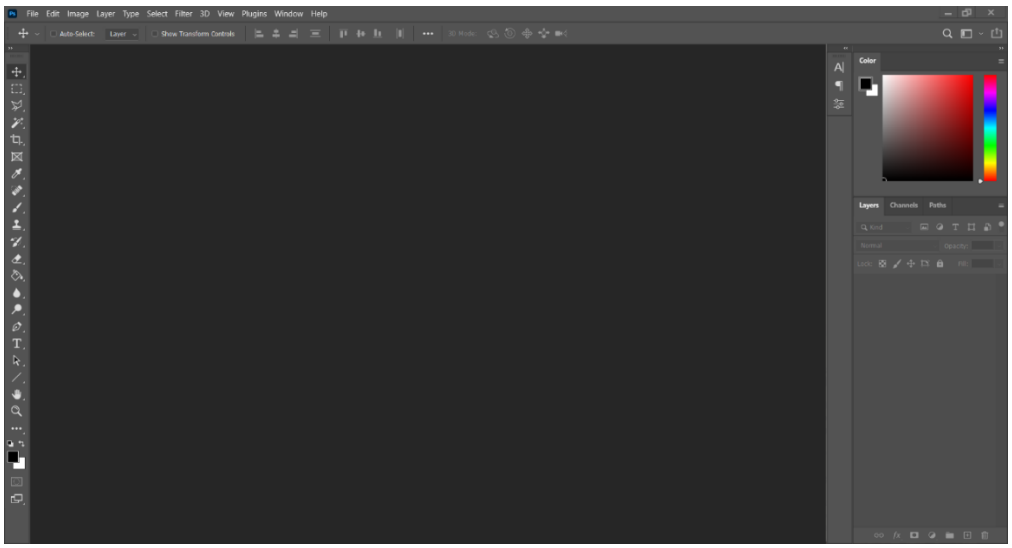
Tutkielman projektissa oli käytössä useita erilaisia ohjelmistoja, joilla toteutettiin peliprojektin eri osa-alueita. Adobe Photoshopilla tehtyjä tuotoksia ovat itse peliprojektissa näkyvät graafiset tuotokset, kuten myös tutkielmaan sisältyvät omat ja mukaillut tuotokset. Tietokonepelin kehitys tapahtui Unity Technologies -yrityksen tarjoamalla Unity -pelimoottorilla. Unity -pelimoottori käyttää C# -ohjelmointikieltä ja tämän takia Unity tarjoaa Microsoft Visual Studio -ohjelman integroinnin. Tutkielman ohjelmistokehitysympäristönä oli tämän takia Microsoftin tarjoama Visual Studio. Projektinhallintaa varten tutkielmassa käytettiin Kanban Tool -projektinhallintapalvelua.

5.1.1 Adobe Photoshop

Adobe Photoshop on Adobe Inc. osakeyhtiön kehittämä ja ylläpitämä kuvankäsittelyohjelma (Adobe, 2021). Adobe Photoshop on osa Adoben tarjoamaa Creative Cloud palvelua, jossa Photoshopin voi ladata Apple iPadille tai pöytäkoneelle kuukausimaksua vastaan. Photoshopin käyttöoikeuden voi tilata erikseen tai osana Adoben tarjoamia ohjelmapaketteja. Tutkielmassa käytössä oli Adobe Photoshop 2021 versio 22.1.1, joka oli tutkielman aikaan Photoshopin uusin version.

Photoshop sai alkunsa Display nimisenä ohjelmana, jonka Thomas Knoll kehitti omaan käyttöönsä harmaasävykuvien toistamiseen yksivärisellä tietokonenäytöllä (Lindblad,

2020). John Knoll kiinnostui veljensä luomasta ohjelmasta ja ehdotti, että Thomas jatkaisi ohjelman kehittämistä kuvankäsittelyohjelmaksi. Yhdessä veljekset jatkoivat ohjelman kehittämistä ja muuttivat ohjelman nimen Photoshopiksi. Vuonna 1988 osakeyhtiö Adobe Inc. kiinnostui veljesten esittelemästä kuvankäsittelyohjelmasta ja veljekset päätyivät hyväksymään Adoben ohjelmasta tekemän tarjouksen (Lindblad, 2020). Adobe julkaisi Photoshopin ensimmäisen version vuonna 1990 ainoastaan Applen Macintosh tietokonealustalle. Photoshopin kahdeksas versio koki tuotemerkkimuutoksen, koska Adobe liitti Photoshopin osaksi Adobe Creative Suite -kokoelmaa. Adobe Creative Suite -ohjelmia pystyivät asiakkaat ostamaan yksittäin tai erilaisissa Adoben tekemissä valmiissa paketeissa (Paananen, 2012). Adobe Creative Suite -kokoelmaan kuuluvat ohjelmat tunnistaa *CS* -tuotemerkistä. Photoshop koki jälleen kerran tuotemerkkimuutoksen Adobe Photoshop CS6 jälkeen, kun Adobe liitti Photoshopin osaksi Adoben tarjoamaan Creative Cloud -palvelua (Dove, 2013). Adobe Creative Cloud muutoksen takia *CS-tuotemerkki* vaihtui *CC* -tuotemerkkiin. Adobe Photoshop 2020 ja uudemmat versiot lopettivat *CC*-tuotemerkin käytön, mutta se ei muuttanut Adobe Creative Cloud toiminnallisuutta millään lailla (Lindblad, 2020). Adobe Photoshop 2020 version mukana Adobe julkaisi myös oman version Apple iPadille, jolla käyttäjät pystyvät muokkaamaan samoja PSD-tiedostoja.



Kuva 2. Adobe Photoshopin käyttöliittymä.

Adobe Photoshop on yksi käytetyimmistä graafisen tuotannon työkaluista. Photoshopin laaja käyttö graafisessa tuotannossa on johtanut ohjelman nimen käyttöön verbinä (Kastrenakes, 2020). Esimerkiksi kuvan muokkausta kuvankäsittelyohjelmalla voidaan yksikertaisesti kutsua sanalla ”photoshopata.” Adoben tuotemerkkiohjesäännöissä kuitenkin todetaan, että Photoshop -tuotemerkkiä ei saa käyttää verbinä (Adobe Inc., 2021a). Tästä huolimatta Photoshopin käyttö verbinä on hyvin yleistä.

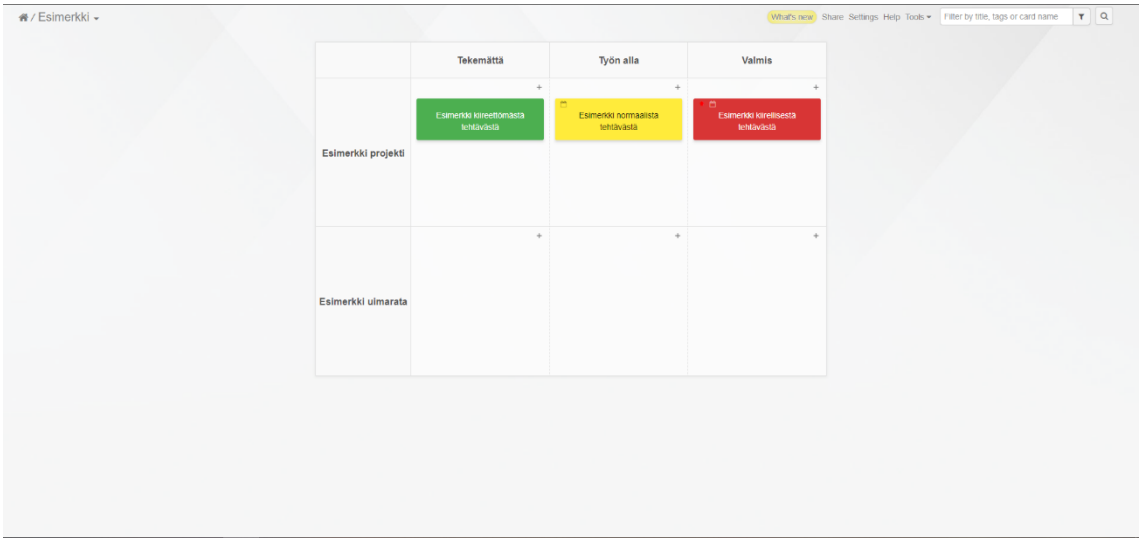
Adobe Photoshop tarjoaa graafisen tuotannon harjoittajille useita erilaisia työkaluja kuvien, videoiden ja 3D-mallien muokkausta varten (Smith, 2020). Photoshop -tiedostojen oletus tallennusmuotoa kutsutaan Photoshop tallennusmuodoksi (PSD). PSD-tiedostoihin voidaan tallentaa kaikki työssä käytetyt toiminnot, kuten esimerkiksi tasot, tehosteet sekä maskit. PSD-tiedostojen maksimi koko on kaksi gigatavua. Tiedoston ollessa suurempi kuin kaksi gigatavua, voi tiedoston tallentaa suuressa tallennusmuodossa (PSB) (Adobe, 2021b).

5.1.2 Kanban Tool

Kanban Tool on Shore Labs yrityksen kehittämä ja tarjoama projektinhallintapalvelu. Shore Labs tarjoaa kolme erilaista versiota Kanban Toolista: *Enterprise, Team* ja *Free* (Kanban Tool, 2021a). Tutkielman käytössä oli *Free* eli ilmainen versio. Ilmainen versio Kanban Toolista tarjoaa käyttäjille kaksi Kanban-taulua, joita voi käyttää korkeintaan kaksi ihmistä. Näihin tauluihin ja ohjauskortteihin ei voi ilmaisversiossa liittää tiedostoja. Kanban Toolista on myös mahdollista asentaa ohjelmistoversio nimeltä Kanban Tool On-Site, joka on kuukausimaksun sijasta vuosimaksulla toimiva tuote (Kanban Tool, 2021b). Ilmaisversio Kanban Toolista sopi tutkielman projektiin hyvin, koska projektissa työntekijöiden määrä oli vain yksi.

Kanban Toolin tarjoamalla Kanban-taululla on oletuksena minimalistinen ulkoasu, mutta sitä voivat käyttäjät muokata joko omilla tai Kanban Toolin tarjoamilla taustakuvilla sekä

teemoilla. Tutkielman kuvankaappauksissa on käytössä Kanban Toolin oletusulkonäkö, joten Kanban-taulun ulkonäköä ei ole kuvankaappauksissa muokattu millään tavalla.



Kuva 3. Kanban Toolin käyttöliittymä.

Kanban Tool ei tutkielman aikaan tarjonnut palveluansa suomenkielisenä. Kuitenkin itse Kanban-taulun ja ohjauskorttien otsikot ja syötekentät käyttäjä pystyy kirjoittamaan suomeksi. Ohjauskortteihin Kanban Tool tarjoaa yksitoista erilaista valmista syötekenttää ja viisitoista käyttäjän muokkausta varten annettua muokattavaa kenttää. Käyttäjät pystyvät myös vaihtamaan ohjauskorttien väriä ja tärkeysjärjestystä.

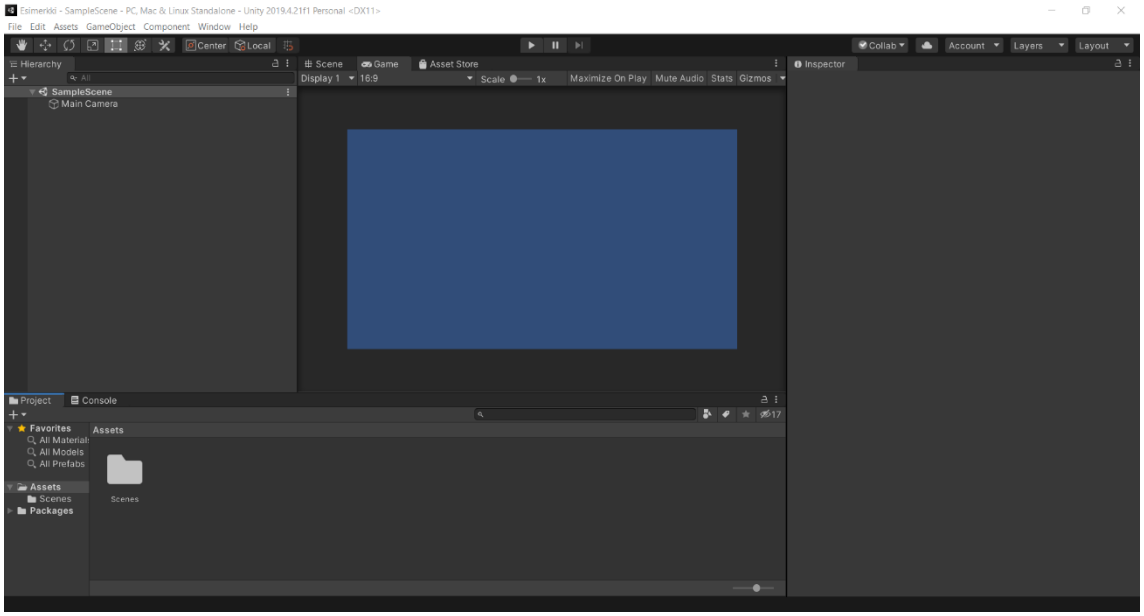
Kanban-taulun sarakkeiden ja uimaratojen määrää pystyy käyttäjä helposti muokkaamaan taulun editorista. Uutta Kanban-taulua luodessa Kanban Tool tarjoaa kolme valmiiksi tehtyä saraketta (*tekemättä, työn alla* ja *valmis*), mutta käyttäjä voi muokata niitä niin kuin parhaaksi näkee. Sarakkeille voi asettaa tehtävärajoituksia, joilla voidaan estää liian monen yhtäaikaisen tehtävien teon. Sarakkeita voi myös jakaa osiin, esimerkiksi *työn alla* -sarake voidaan jakaa *kehitys*- ja *testaus* -osioihin.

Kanban Tool tarjoaa myös Kanban-tauluun lisäosia nimellä *Power-Ups*. Nämä lisäosat voivat olla esimerkiksi erilaisia visuaalisia tehosteita korteille tai vaikka kalenteripienoisohjelma.

5.1.3 Unity

Unity on Unity Technologiesin kehittämä ja tarjoama pelimoottori. Harrastajat voivat ladata ja käyttää *Unity Personal* -versiota ilmaiseksi, kunhan harrastajien liikevaihto Unityllä tehdyistä tuotteista ei ylitä 100 000 dollaria vuodessa. Pienyritykset voivat myös käyttää *Personal* -versiota ilmaiseksi, mutta yritysten koko liikevaihto ei saa ylittää samaa sadantuhannen dollarin rajaa (Unity Technologies, 2021a). Maksuttoman version lisäksi Unity Technologies tarjoaa *Plus-, Pro-* ja *Enterprise* -versiot, jotka tarjoavat laajemman valikoiman kehitystyökaluja.

Unity on yksi käytetyimmistä pelinkehitysalustoista ja on osoittautunut olevan todella suosittu indie-videopelikehittäjien keskuudessa. Esimerkiksi Tea for Two -pelistudion perustajajäsen Garry Williams vitsaili haastattelussa, kuinka 99 % indie-pelistudioista luultavasti käyttää Unity pelimoottoria Eurogamer Expo (EGX) messuilla (Dealessandri, 2020). Unityn tarjoama ilmainen lisenssi pienyrityksille ja harrastelijoille on mahdollistanut pelinkehitysalustan laajan leviämisen. Tämä on johtanut useiden ilmaisten oppaiden ja kurssien luomiseen, jotka ovat helpottaneet Unityn valitsemista pelinkehitysalustana. Vuonna 2019 Unity Technologies hankki oikeuden Ludiq työkalukehittäjän luomaan visuaaliseen Bolt -skriptaustyökaluun (Dealessandri, 2020). Bolt mahdollistaa logiikan luomisen peleihin ja ohjelmiin ilman koodausta. Vuonna 2020 Unity Technologies lisäsi Boltin osaksi Unity kehitysalustaa, ja sen voi ladata ilmaiseksi Unityn *Asset Storesta* (Unity Technologies, 2021b). Boltin lisäys kehitysalustalle alentaa ohjelmistokehityksen alalle pääsemisen esteitä ja tarjoaa nopean ja helpon keinon luoda ohjelmistologiikkaa sekä prototyyppejä.



Kuva 4. Unityn käyttöliittymä.

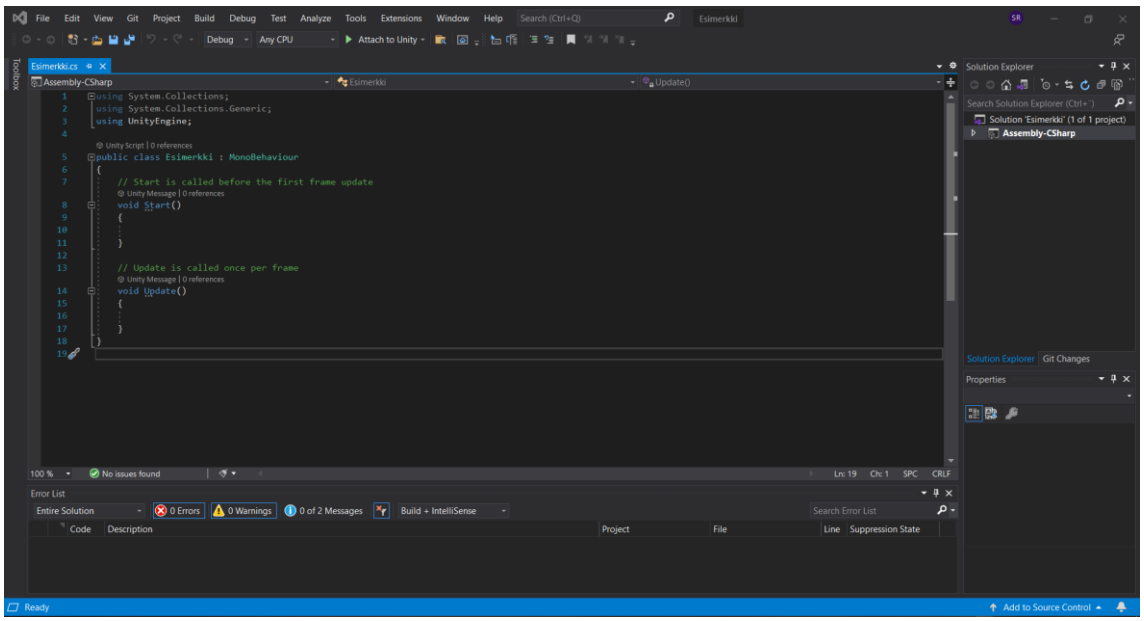
Unity kehitysalustaa voidaan myös käyttää pelinkehityksen ulkopuolella. Unityä on esimerkiksi käytetty Disneyn Leijonakuningas (2019) elokuvan kehityksessä ja erilaisissa simulaatio tehtävissä. Unity Technologies raportoi, että noin 10 % yrityksen liikevaihdosta tapahtuu pelinkehityksen ulkopuolella (De Vynck, 2020).

Unity käyttää C# -ohjelmointikieltä. Unityn asennuksen mukaan on liitetty Microsoft Visual Studio ohjelmointiympäristö C# ohjelmointia varten. Käyttäjillä on myös mahdollisuus käyttää muitakin ohjelmointiympäristöjä, kuten esimerkiksi JetBrains yrityksen tarjoamaa Rider -ohjelmointiympäristöä (JetBrains, 2021). Unityllä on myös mahdollista käyttää muitakin ohjelmointikieliä, mutta muiden kielien käyttö vaatii kolmannen osapuolen tarjoamien ohjelmistojen asennuksen.

5.1.4 Visual Studio

Visual Studio on Microsoft Corporationin kehittämä ja tarjoama ohjelmointiympäristö, joka tukee useita eri ohjelmointikieliä. Harrastajat voivat ladata ja käyttää *Visual Studio Community* -versiota ilmaiseksi. Pienyritykset, joissa on maksimissaan viisi yhtäaikaista

Visual Studion käyttäjää voivat myös käyttää *Community* -versiota ilmaiseksi. (Microsoft, 2021a). Maksuttoman version lisäksi Microsoft tarjoaa *Professional*- ja *Enterprise* -versiot, jotka tarjoavat laajemman valikoiman kehitystyökaluja.



Kuva 5. Microsoft Visual Studion käyttöliittymä.

Visual Studio tarjoaa älykkään koodin täydennysominaisuuden nimeltä IntelliSense. IntelliSense avustaa käyttäjää tarjoamalla koodin täydennyksiä ja tietoja kuinka erilaisia parametrejä voidaan koodissa käyttää. Microsoft on luonut IntelliSensen tueksi koneoppimiseen pohjautuvan IntelliCode -järjestelmän. IntelliCode antaa käyttäjälle koneoppimiseen pohjautuvia täydennysohjeita, jotka voivat toimia myös projektin jäsenten välillä (Microsoft, 2021b). Ryhmätyöskentelyssä voi myös koneoppimisen sijasta käyttää ohjelmavarastoa, mutta ryhmällä täytyy silloin olla käytössä Git -versionhallintajärjestelmä. Visual Studion voi liittää osaksi Unityä, luoden saumattoman ohjelmointiympäristön. Liitoksen avulla Visual Studion IntelliSense pystyy tarjoamaa käyttäjälle Unityn ohjelmointirajapinnan täydennyksiä ja niiden toimintoja.

5.2 Breakout

Tutkielman projektin toteutuksen kohteena oli Atarin vuonna 1978 tuottama ja julkaisema *Breakout* -videopeli Atari 2600 -videopelikonsolille. Breakout -videopelissä pelaaja ohjaa mailaa, joka voi liikkua vain x-akselilla. Mailan avulla pelaaja pystyy pompottaa pelissä olevaa palloa, jolla pelaajan on tarkoitus hajottaa mailan yläpuolella olevaa tiiliseinää. Pelaajalla on käytössä Atari 2600 -videopelikonsoliversiossa viisi elämää tai yritystä. Pallon koskettaessa näytön alareunaa pelaajaa menettää yhden yrityksen ja yritysten loppuessa peli päättyy. Peli asettaa pelaajalle uuden ehjän tiiliseinän, jos pelaaja onnistuu hajottamaan koko tiiliseinän. Pelin vaikeustaso kasvaa pelin aikana pelaajan hajottaessa tiiliseinästä tiiliä. Vaikeustason kasvu ilmentyy pelissä pallon nopeuden kasvuna, joka myös lisää tiileistä saatavien pisteiden määrää.

Kuva 6. Atari 2600 Breakout -peli (Oberon Gaming, 2019).

Tutkielman tutkimuksen aiheena oli tarkastella Kanbanin soveltuvuutta ohjelmistokehityksessä ketteränä projektinhallintamenetelmänä. Tutkimuksessa tuotettavan peliaiheen valinnaksi suunniteltiin jo olemassa oleva peli. Oman uuden peli-idean suunnittelu olisi mahdollisesti voinut kuluttaa Kanbanin tarkasteluun varattua aikaa ja johtaa

tutkimuksen laajuuden lipsumiseen. Tämän takia tutkimuksessa oli tärkeää valita tutkimuksen laajuuteen sopiva peliaihe.

Ensimmäinen mahdollinen peliaihe oli Atarin Pong -peli, mutta Pong todettiin olevan liian yksinkertainen toteuttaa. Pong -pelissä tärkeimmät kehityksen alueet olivat mailan ja pallon liike sekä pistenäkymä. Breakout valittiin tutkimuksessa kehitettäväksi peliksi, koska Breakout -pelissä oli Pongiin verrattuna enemmän kehityksen alueita. Breakout ei kuitenkaan ollut liian yksinkertainen pelivalinta, koska peliin piti luoda moniosainen tiiliseinämä, pallon ja mailan liike sekä pistenäkymä. Näiden lisäksi tuotettavaan peliin suunniteltiin lisättävän päävalikko sekä loppunäkymä. Uusien näkymien suunniteltiin lisäävän enemmän visuaalista kehitystyötä tuotettavaan peliin.

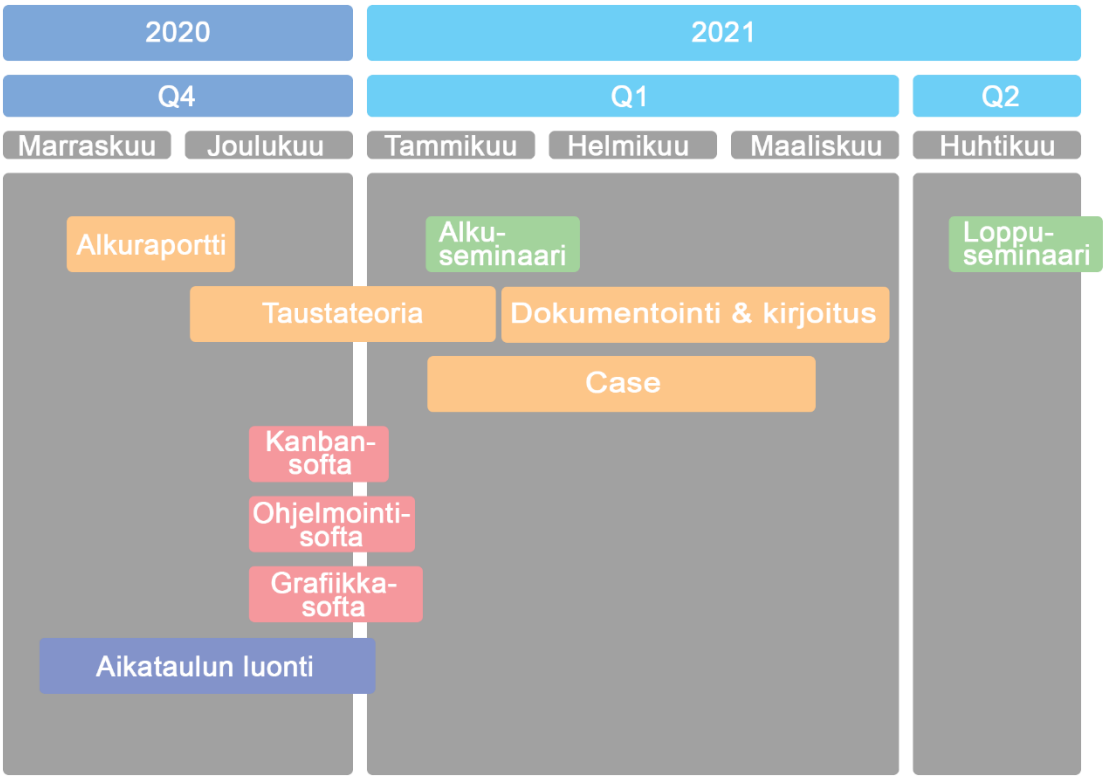
5.3 Tietokonepeliprojekti

Tämä luku käsittelee tietokonepelin kehitysprosessin kulkua ja Kanbanin käyttöä projektin hallinnassa. Projektista saadut tärkeimmät havainnot ovat luvussa esitelty kuvankaappauksien avulla, jotta saadut tulokset olisivat näkyvästi esitettynä. Kehitysprosessista saadut havainnot muodostavat pohjan tutkielman fenomenologiselle analyysille.

5.3.1 Suunnitelma

Tietokonepeliprojekti alkoi suunnitelmasta, jossa määriteltiin pelin aihe, projektissa tarvittavat ohjelmat ja työkalut sekä aikatalutus. Pelin aiheeksi valittiin Atarin Breakout -peli ja vielä tarkemmin määriteltynä Atari 2600 -konsoliversio pelistä. Breakout soveltui tutkielman tavoitteeseen olemalla melko lyhyt ja yksinkertainen peli. Peli ei kuitenkaan ollut liian yksinkertainen, jotta projektissa voitiin tutustua pelinkehityksen eri osa-alueisiin. Pelissä pääoliot ovat: maila, pallo, tiilet, pisteet ja elämät. Projektin kulkua johti näiden olioiden tuottaminen Kanban-taulussa olevien ohjauskorttien avulla.

Pelin aiheen suunnittelemisen jälkeen oli vuorossa alustavan aikataulun luonti, jotta saatiin määriteltyä projektin kesto ja tarpeet. Projektinhallinnan teorian opettelu ja tutkiminen oli tärkeä ensimmäinen vaihe aikataulussa tutkielman alkuraporttia ja projektissa käytettävien ohjelmien valintaa varten. Teoriassa ilmeni Scrumin sekä Scrumban laaja käyttäjäkunta Kanban -menetelmään verrattuna. Tämän takia tutkielmassa oli tärkeää tarkastella muidenkin projektinhallintamenetelmien toimintoja teoriaosiossa.

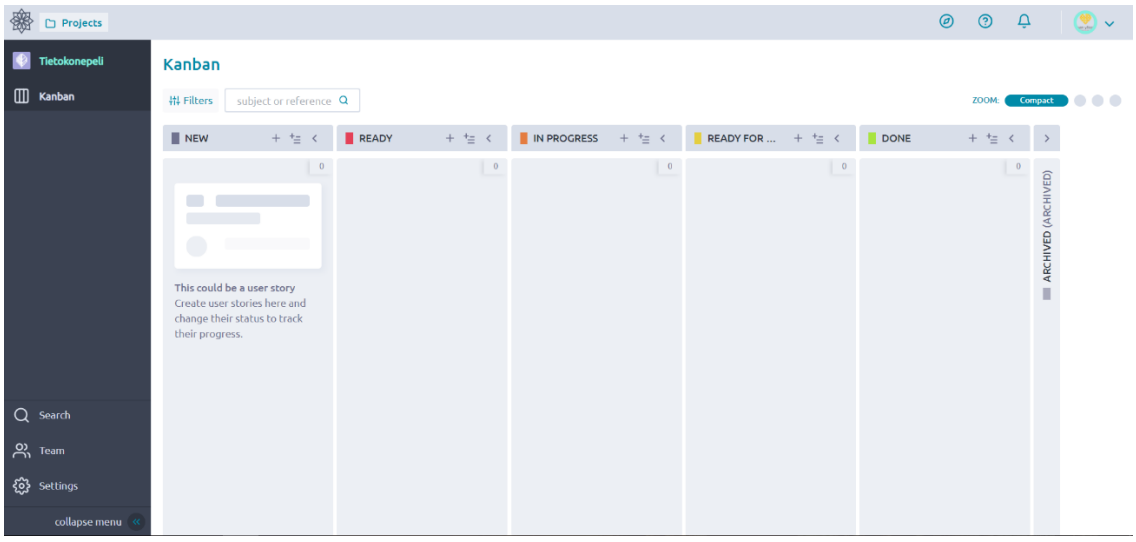


Kuvio 10. Suunniteltu aikataulu tutkielman työstämiselle.

Kanban projektinhallintaohjelman valinnassa oli teorian opettelun jälkeen mukana ajatus ohjelman toimivuudesta myös Scrumin käytössä. Pitäisikö tutkielmassa käytetyn ohjelman olla kykenevä siirtämään projektin tietoja eri hallintamenetelmien välillä? Tämän takia ensimmäinen tutkielmassa harkittua Kanbania käyttävä ohjelma oli Atlassian yrityksen tarjoama Trello -palvelu. Palvelun testaus jäi kuitenkin tekemättä, koska palvelu pyysi maksutietoja ennen kuin palvelun ilmaista versiota pääsee käyttämään. Tämä

kuitenkin myöhemmin osoittautui epätodeksi ja maksutietojen antosivulla oli tietojen alapuolella pieni näppäin, jossa käyttäjä pystyi kirjautumaan ilman maksutietojen antamista. Tästä maksutietosivusta aiheutuneen erehdyksen takia Trellon valinta projektinhallintaohjelmana jäi tekemättä.

Trellon jälkeen testauksen kohteeksi valittiin Taiga Agile yrityksen julkaisema avoimen lähdekoodin projektinhallintaohjelma Taiga. Taigan ulkonäkö on yksinkertainen sekä selkeä ja projektin luonti Taigalla oli helppoa. Ensimmäinen ensivaikutusta huonontava ongelma kuitenkin ilmaantui, kun käyttöliittymän kielen vaihtoi suomen kieleen. Suomenkielisissä teksteissä oli useita käännösvirheitä ja tämän takia suomen kielen käyttö ei ollut vakuuttavaa. Toinen ongelma oli Kanban-taulun ohjauskorttien hallinnan toiminta, joka toimi käyttäjän ajatusmallia vastaan. Ajatusmallia kutsutaan englanninkielisellä termillä *”mental model”.* Sillä tarkoitetaan käyttäjän oletusta siitä, kuinka ohjelma ja sen toiminnot käyttäjän mielestä pitäisi toimia. Taigassa ohjauskortteja ei voi avata tai muokata suoraan taulunäkymässä, vaan käyttäjän täytyy avata projektinäkymä valiten sieltä haluttu ohjauskortti. Käyttäjän ajatusmallia vastainen toiminto johti kolmannen mahdollisen Kanban -ohjelman testaamiseen.



Kuva 7. Taigan käyttöliittymä.

Kolmannes projektinhallintaohjelma oli Kanban Tool. Kanban Tool on Shore Labs yrityksen kehittämä ja tarjoama projektinhallintapalvelu, jonka taulun ulkonäkö on hyvin yksinkertainen. Taulua voi käyttäjät kuitenkin muokata oman halunsa mukaan. Kanban Toolin ohjauskortteja voi avata painamalla korttia taulussa. Ohjauskortin painaminen taulussa avaa tarkemman näkymän ohjauskorttien tiedoista. Ohjelman yksinkertaisuus ja helppotoimivuus johti sen valintaan tutkielman projektinhallinnantyökaluna.

Projektinhallintatyökalun valinnan jälkeen vuorossa oli työtehtävien määrittely ja niiden suorittamiseen käytettävät ohjelmat. Työssä käytetyn pelimoottorin valinta tapahtui kahden eri pelimoottorin välillä, eli valintana oli joka Unity tai Unreal Engine. Unity valittiin projektissa käytettäväksi pelimoottoriksi, koska Unityssä ohjelmointi kielenä on käytössä entuudestaan tuttu C#. Unity käyttää myös entuudestaan tuttua Microsoft Visual Studio -ohjelmointiympäristöä, joten pelimoottorin valinta oli melko selkeä. Pelin ja tutkielman visuaalisuuden tuottamiseen valittiin Adobe Photoshop, koska ohjelma oli Visual Studion tapaan jo tutuksi tullut. Photoshop on ohjelma, joka oli myös valmiiksi asennettuna työssä käytössä olleilla tietokoneilla.

5.3.2 Ohjelmistojen asennus ja käyttöönotto

Suunnitelmien ja alustavien työtehtävien määrittelyn jälkeen oli aika aloittaa ensimmäisten tehtävien toteutus, eli ohjelmistojen asennus sekä niiden käytön opettelu. Kanban Tool on internetin välityksellä toimiva palvelu, joten sitä ei tarvinnut asentaa erikseen, vaan projektia sai heti käyttäjätunnuksen luotua aloittaa tekemään. Ensimmäisenä asetettiin projektille nimi ja tässä tapauksessa projektia kutsuttiin nimellä: ”Tietokonepelin kehitys.” Projektin nimen valinnan jälkeen oli vuorossa työvirran sarakkeiden määrän ja tarkoitusten valinta. Kanban tool tarjosi automaattisesti Kanban-taulun kolme pääsaraketta, mutta käyttäjä sai muokata niitä tahtonsa mukaan. Tutkielman projektissa sarakkeina olivat *tehtävät, tekeillä* ja *tehty*. *Tekeillä* -sarake oli jaettu kahteen osaan *kehitys* ja *testaus*. *Tekeillä* -sarakkeeseen asetettiin myös tehtävärajoitus, joka rajoitti sarakkeessa yhtä aikaa tehtävien töiden määrää kahteen. Rajoituksen tarkoituksena oli mahdollistaa

työnteon ja testauksen samanaikaisuuden ilman, että työtä olisi samanaikaisesti liikaa. Sarakkeiden luonnin jälkeen määriteltiin ohjauskorttien kiireellisyydet, tarkoitukset sekä niiden värit. Kanban Tool ei ohjauskorttien alkumäärittelyn aikana antanut muokata korttien sisältöä, joka nopeutti Kanban-taulun luontia. Ohjauskorttien luonnin jälkeen Kanban-taulu oli valmis käytettäväksi. Ensimmäiseksi muokattiin projektin ohjauskorttien sisältöä. Kanban Tool tarjosi useita erilaisia valmiita työkenttiä ohjauskortteihin, mutta käyttäjille oli myös annettu viisitoista omaa muokattavaa kenttää.

Projektissa käytettyjen ohjauskorttien sisältö:

1. **Description**

   Ohjauskorttien tehtävien kuvaus. Kuvauksessa on tarkoitus kirjoittaa tehtävän tarkempi kuvaus, koska tehtävän otsikko ei välttämättä riitä kertomaan kaikki tehtävään liittyvät asiat.

2. **Card type**

   Ohjauskortin luokalla määritellään kortin tehtävän tarkoitusta antamalla kortille oman värinsä. Esimerkiksi testauksen tehtävä voi olla sininen kortti ja koodaamisen tehtävä punainen kortti.

3. **Priority**

   Ohjauskorttien tärkeyttä kuvaava kenttä. Kiireellinen tehtävä saa korttiin punaisen nuolen, kun taas alhaisen tärkeyden tehtävä saa vihreän nuolen. Tavalliset tehtävät eivät saa mitään näkyvää erikoismerkintää Kanban-taulussa.

4. **Due date**

   Ohjauskorttien tehtäville annettu määräpäivä, jolloin tehtävän on oltava valmis. Määräpäivä -kenttää painettaessa käyttäjälle ilmestyy kalenterinäkymä, jolla annetaan tehtävälle määräpäivä.

5. **Assignment**

   Ohjauskorttien työnjakokentällä määritellään kuka tai ketkä tekevät ohjauskortin tehtävän.

6. **Estimated time**

   Ohjauskorttien arvioidulla aikakentällä voidaan määritellä arvioita aika tehtävän kestoon. Sillä voidaan myös osoittaa, että tehtävään on varattu vain sen verran aikaa.

7. **Tags**

   Ohjauskorttien tunnisteilla voidaan määritellä tehtävä tiettyyn luokkaan. Tehtävillä voi olla useita tunnisteita. Tutkielman projektissa oli käytössä tunnisteet: *koodaus, ohjelmisto, ongelma, teksturointi, testaus* ja *ääni.*

8. **Checklist**

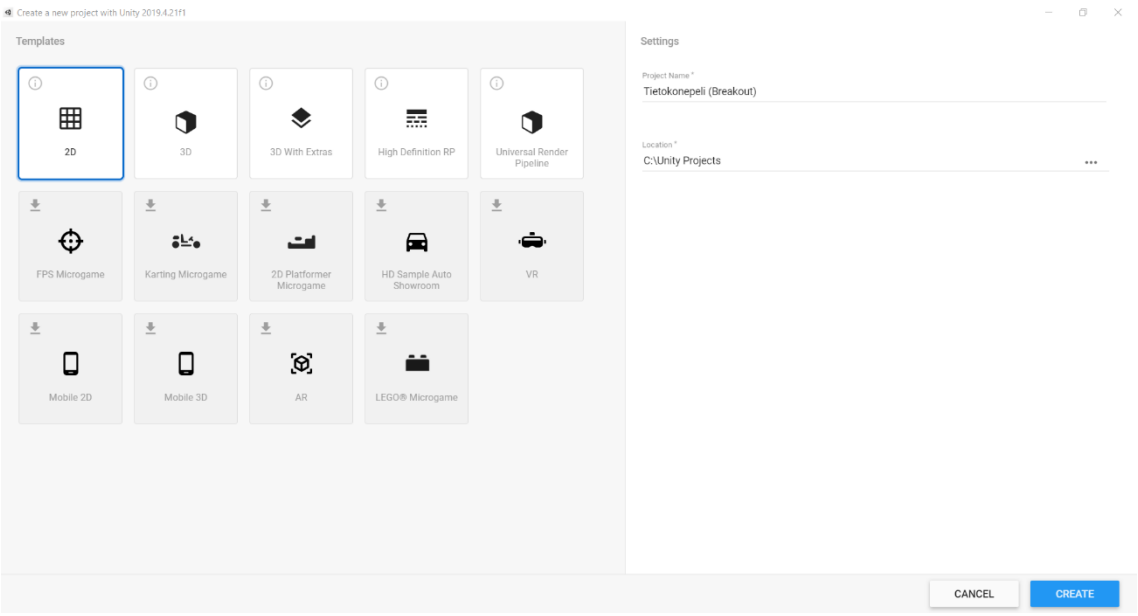
   Kanban Toolin -ohjauskortteihin on myös mahdollista asettaa muistilista, jota käyttäjät voivat täyttää tehtävän kohteiden valmistuessa.

9. **Custom Field 1 (Comments)**

   Yksi käyttäjille annetuista muokattavista kentistä. Jokaisella Kanban Toolin -ohjauskortilla on oma kommentointiosio, mutta kuvankaappausta varten kommentointi on liitetty osaksi itse tehtäväkorttia.

Tutkielman projekti oli melko yksinkertainen, joten korttien luokkien väri määriteltiin niiden tärkeyden mukaan, eikä itse tehtäväluokan mukaan. Tämä helpotti kuvankaappausten ottamista, koska Kanban Toolissa tehtävien tärkeys on merkittynä kuvankaappauksessa melko hankalasti havaittavalla pienellä nuolella. Tämän takia tutkielman projektin ohjauskorteissa oli käytössä kolme tehtäväluokkaa, jotka olivat määritelty tehtävän tärkeyden mukaan: *alhainen tärkeys* (vihreä)*, oletus* (keltainen) ja *kiireellinen* (punainen).

Unityn asennus oli mutkatonta ja asennuksen aikana kävi ilmi, että Visual Studion asennus tapahtui samalla kerralla. Käyttäjän ei tarvinnut miettiä Visual Studion liittämistä Unityyn ollenkaan, koska liitokset olivat suoraan tarjottu käyttäjälle. Uutta Unity -projektia luodessa, Unity tarjoaa käyttäjälle valmiita ohjelmointipohjia. Breakout on kaksiulotteinen peli, joten projektinpohjaksi valittiin 2D-pohja. 2D-pohja muutti tiedoston asetukset valmiiksi kaksiulotteista projektia varten. Käytännössä tämä tarkoitti, että esimerkiksi projektin oletusnäkymä oli kaksiulotteisilla asetuksilla ja projekti oli valmis käyttämään sprite-grafiikkaa eli pikseligrafiikkaa.



Kuva 8. Unityn projektinluonninnäkymä.

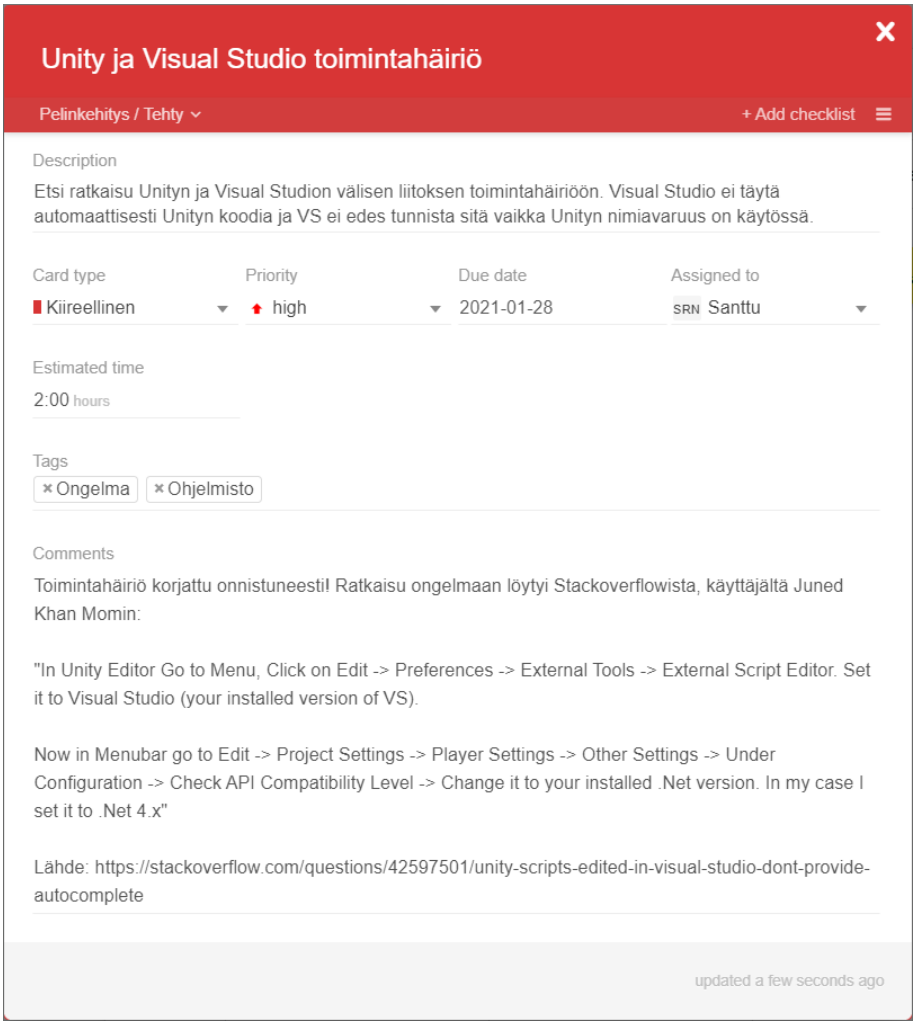
Peliprojektin tiedoston alulle saannin jälkeen projektissa oli vuorossa Unityn -käyttöliittymään tutustuminen. Ensimmäiseksi piti selvittää kuinka lisätä visuaalisia olioita peliympäristöön ja kuinka ohjelmoida niille toimintoja. Adobe Photoshopilla luotiin yksinkertaiset maila-, pallo- ja tiilikuvat, joille oli seuraavaksi vuorossa toimintojen ohjelmoiminen.

5.3.3 Ohjelmointi

Peliprojektin ohjelmointi alkoi lisäämällä pallokuva Unityn -peliympäristöön. Tämän jälkeen pallolle lisättiin *Box Collider 2D* -komponentti eli pallolle annettiin kaksiulotteinen laatikonmuotoinen törmäysalue. Törmäysalue voi olla suurempi tai pienempi kuin itse käytössä olevan kuvan alue, mutta oletuksena alue mukautuu kuvan kokoon. Ilman törmäysalueen määrittelyä pallo liikkuisi pelin sisällä mailan ja tiilien lävitse. Pallolle seuraavaksi asetettu komponentti oli *Rigidbody 2D* eli jäykkä kaksiulotteinen runko. Rungolla voidaan lukita pallon liike x- ja y-akseleilla sekä pallolle voidaan asettaa erilaisia painovoimaan ja massaan liittyviä tietoja. Pallolle asetettiin pyörimistä estävä valinta, koska Breakout -pelissä oleva nelikulmioinen pallo ei pyöri. Kaksiulotteisessa rungossa painovoiman vaikutus on oletuksena asetettu päälle, mutta tässä peliprojektissa sitä ei tarvittu. Tämän jälkeen luotiin uusi C# -skriptitiedosto ja sille annettiin nimike PallonLiike. Uusi tiedosto lisättiin palloon komponenttina ja lisäämisen jälkeen oli aika aloittaa pallon toimintojen ohjelmoiminen.

C# -skriptitiedoston avaaminen käynnisti automaattisesti Unityyn liitetyn Microsoft Visual Studion. Luotuun tiedostoon oli valmiiksi lisätty UnityEngine -nimiavaruus ja muutamia ohjeita antava koodinpätkä. Ohjelmoinnin alussa tuli kuitenkin hyvin nopeasti vastaan ongelma nimiavaruuksien kanssa. Visual Studio ei tarjonnut Unityssä käytetylle koodille automaattista täydennystä, eikä edes tunnistanut koodia. Tämä ongelma johti ensimmäiseen projektin suunnitelmista poikkeavan tehtävän luomiseen. Uusi ongelmaan liittyvä ohjauskortti lisättiin Kanban-tauluun nimikkeellä ”Unity ja Visual Studio toimintahäiriö”. Kyseessä oli projektin suorittamista estävä ongelma, joten ohjauskortti sai kiireellisen luokituksen. Edeltävä pallon liikkeeseen liittyvä ohjauskortti siirrettiin takaisin kehityssarakkeesta tehtäväsarakkeeseen. Ratkaisu ongelmaan löytyi lopulta Stack Overflow -verkkosivustolta. Stack Overflow on pääasiallisesti ohjelmointiin liittyvien ongelmien ja kysymyksien ratkaisua varten luotu kysymys-vastaus-sivusto (Q&A). Ongelmaan oli annettu monia eri vastauksia, mutta jokaisessa vastauksessa ongelman lähde oli sama. Visual Studiossa koodausprojektin vasemmassa ylälaidassa pitäisi lukea ”Assembly-CSharp”, mutta ongelmatapauksen kohdassa luki ”Miscellaneous Files”. Unity kokoaa

oletuksena melkein kaikki pelissä käytettävät koodit C# -koodikirjastoon nimeltä Assembly-CSsharp.dll (Unity Technologies, 2021c). Ongelmassa viittaus tähän kirjastoon oli kateissa, joka johti Visual Studion puutteelliseen täydennykseen. Ratkaisu projektin ongelmaan oli manuaalisesti muuttaa Unityssä muutama Unityn ja Visual Studion väliseen liitokseen liittyvä asetus.



Kuva 9. Kanban Toolin ohjauskortti, jossa kuvataan kiireellistä ongelmatehtävää.

Ongelman ratkaisun jälkeen Visual Studio tarjosi puuttuvia täydennyksiä ja itse pallon liikkeen ohjelmointi voitiin aloittaa. Pallolle määriteltiin aluksi käytettävä pelialue. Ilman käytettävän alueen määrittelyä pallo voi lentää kentän ulkopuolelle jatkaen matkaansa loputtomasti ilman painovoiman vaikutusta. Pelialueen rajauksen lisäksi pallolle

määriteltiin tasainen nopeus käyttämällä tietuetta Vector2. Vector2 kuvaa kaksiulotteisia vektoreita ja pisteitä, jotka sopivat projektissa pallon kaksiulotteiseen liikkeeseen.

```
// Äärirajat, joista pallo kimpoilee. Jos numerot ovat isompia
kuin pelille annettu pohja, pallo lentää näytön ulkopuolelle
// Jos rajat ovat pienempiä kuin pelille annettu pohja, pallo
kimpoilee "näkymättömien seinien" välillä
     readonly float vasenOikea = 7.39f;
     readonly float ylaAla = 4.8f;
// Pallon nopeus. Sitä suurempi numero, sen nopeampi pallo
// Tietoa Vector2 https://docs.unity3d.com/ScriptReference/Vec-
tor2.html
     Vector2 nopeus = new Vector2(8, -8);
```

Seuraavaksi vuorossa oli pallon liikkeen ohjelmoiminen ja pallon käyttäytyminen osuessa pelialueen reunoihin. Pallon liikkeen ohjelmoimisen avuksi käytettiin tietuetta Vector3 ja deltaTime float-arvoa. Delta-ajan ja pallon nopeuden avulla voitiin laskea pallon etäisyys ja saada selville pallon sijainti. Pallon sijainnin määrittely oli tärkeää pallon liikkeen ohjelmointia varten seinään tai mailaan törmätessä.

```
// Pallon sijaintien määrittely, eli kuinka pallo kimpoilee reu-
nasta toiseen
// Tietoa Vector3 https://docs.unity3d.com/ScriptReference/Vec-
tor3.html
// Tietoa delta ajasta https://docs.unity3d.com/ScriptRefe-
rence/Time-deltaTime.html
     Vector3 delta = nopeus * Time.deltaTime;
     Vector3 uusiSijainti = transform.position + delta;
```

Pallon liikkeen ohjelmoimisen jälkeen testattiin, että pallo liikkuu näytöllä niin kuin pitääkin. Tämän jälkeen oli vuorossa mailan liikkeen ohjelmointi. Mailan liikkeelle oli suunniteltu käytettävän joko tietokoneen nuolinäppäimiä tai hiirtä. Hiiri valittiin käyttäjän syötevälineeksi, koska hiiren liikkeellä voidaan ohjata mailaa nopeasti ja tarkasti. Mailan liikettä varten luotiin MailanLiike -skriptitiedosto ja mailalle lisättiin törmäyskomponentti sekä jäykkärunko. Rungon avulla mailan y-akselin liike lukittiin, koska mailan oli tarkoitus liikkua vain vasemmalta oikealle. Mailan painovoiman vaikutus asetettiin pois päältä pallon tapaan. Tämän lisäksi mailalle annettiin Unityssä oleva ”Player” eli pelaajatunniste. Mailaan viittaus koodissa voi tämän jälkeen tapahtua käyttämällä pelaajatunnistetta.

Mailalle ohjelmoitiin pallon tapaan käytössä olevat pelialueen rajat ja mailan nopeus. Tämän jälkeen luotiin koodi, joka ottaa hiiren syötteen vastaan ja määriteltiin tämän avulla mailan sijainti pelialueella.

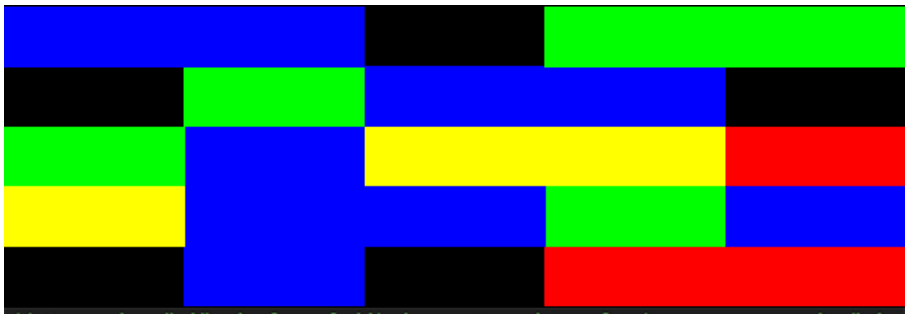
```
void Update()
{
      Transform maila = GetComponent<Transform> ();

      // Otetaan hiiren x-akselin liikkeen syöte vastaan
      float hiiriX = Input.GetAxis("Mouse X");
      maila.position = maila.position + (Vector3.right * hiiriX *
      nopeus * Time.deltaTime);

      // Mailan sijainti
      float mailaX = maila.position.x;
      mailaX = Mathf.Clamp(mailaX, -reunaX, reunaX);
      maila.position = new Vector3(mailaX, maila.position.y,
      maila.position.z);
}
```

Ohjelma 1. Hiirestä saatavan syötteen otto ja käyttö mailan sijainnin määrittämiseen.

Mailan liikkeen ohjelmoimisen ja testauksen jälkeen lisättiin vielä pallon ja mailan törmäykseen liittyvät koodinpätkät. Mailan ja pallon liikkeen toimiessa oli seuraavana vuorossa pelin tiiliseinän luominen.

Tiiliseinää varten luotiin Photoshopilla mailan kaltainen suorakulmainen kuvio, josta lopullinen seinämä rakentui. Luotu tiilikuva lisättiin Unityn peliympäristöön, mutta tiili sijoitettiin pelattavan alueen ulkopuolelle. Tiilille lisättiin pallon ja mailan tapaan törmäysalue, jotta saadaan luotua pallon ja tiilien välinen vuorovaikutus. Seuraavaksi luotiin uusi tyhjä peliolio, jolle annettiin nimi Tiiliseina. Tämän jälkeen luotiin uusi C# -skriptitiedosto, jolla annettiin nimi Tiilet. Luotu uusi skriptitiedosto liitettiin tämän jälkeen Tiiliseina -peliolioon. Pelin tiiliseinää varten täytyi luoda tiileille omat värit. Breakout -pelissä tiilien väritys on sateenkaaren kaltainen eli seinämän riveillä on omat värinsä. Tässä vaiheessa pelinkehitystä oli kuitenkin ajatuksena testata sattumanvaraisen värityksen asettamista tiiliseinälle. Tiiliseinälle annettiin viisi eri väriä, joista oranssi piti määritellä koodissa itse. Värit asetettiin taulukkoon ja luotiin koodi, joka asettaa annetut värit tiileille

sattumanvaraisesti. Uuden luodun tiiliseinän toimivuuden testauksen aikana tuli kuitenkin ilmi, että osa tiilistä oli läpinäkyviä. Läpinäkyvillä tiileillä oli törmäysalueet, joten pallon ja näkymättömien tiilien vuorovaikutus oli kuitenkin toimiva. Tämä johti toiseen odottamattoman ongelmatehtävän lisäämiseen Kanban-tauluun. Ongelman ratkaisussa kesti huomattavasti odotettua kauemmin, vaikka ongelman aiheuttaja oli melko yksinkertainen. Tiiliseinälle luodussa väritaulukossa oli kirjoitusvirhe ja numero kolmosen sijasta taulukossa esiintyi numero kaksi toistuvasti. Tästä pienestä virheestä johtuen osa tiileistä tiiliseinässä oli läpinäkymättömiä. Ongelma olisi voinut ratketa nopeammin, jos sattumanvaraisen värityksen sijasta käytössä olisi ollut sateenkaaren kaltainen värijärjestely. Tasaisissa väririveissä olisi nopeasti tullut selville, että yksi kokonainen rivi oli läpinäkyvä.



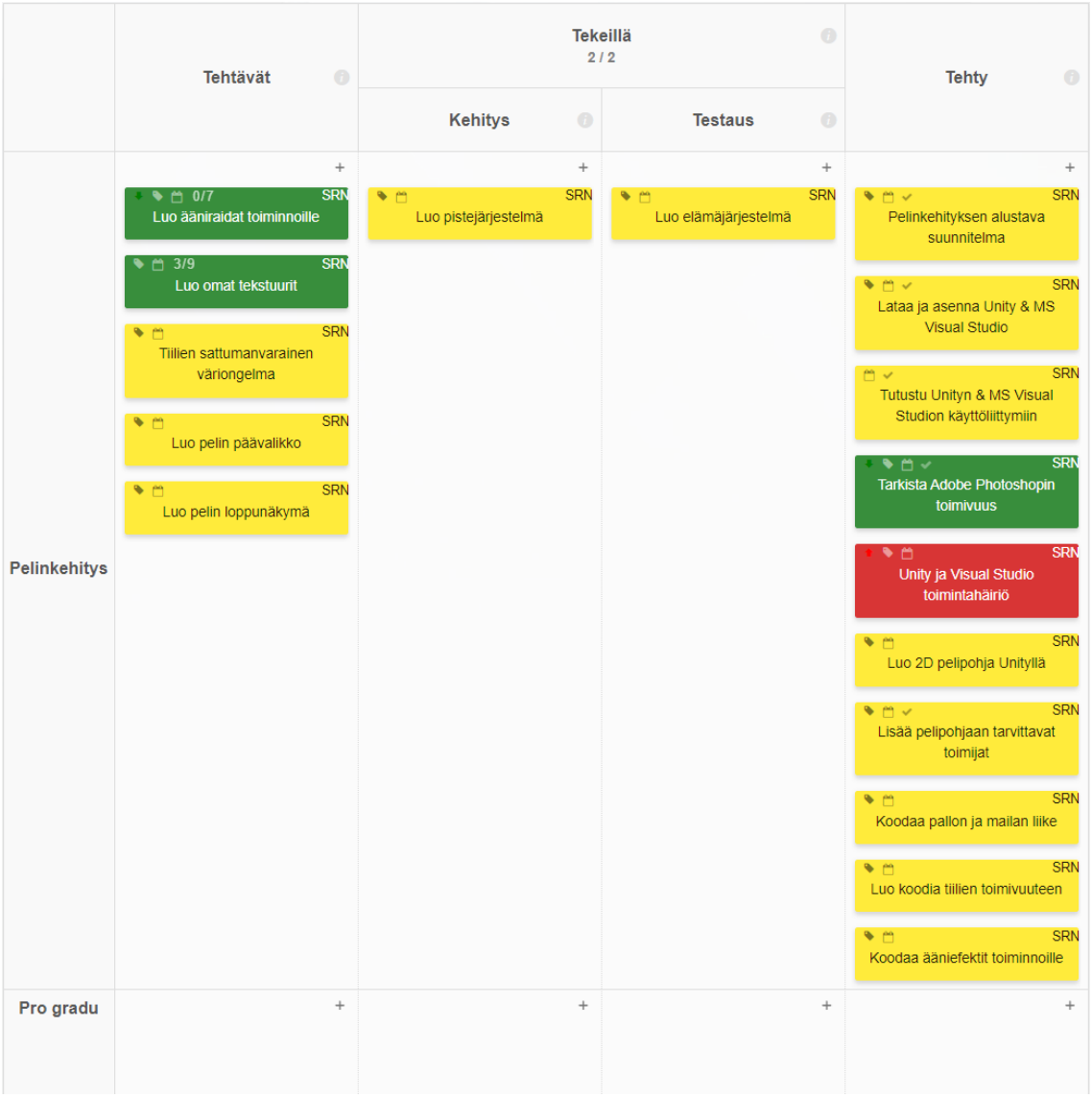
```
// Oranssia väriä ei ole valmiiksi annettu, joten luodaan oma oranssi väri
Color orange = new Color(255, 165, 0);

// Taulukon värit
varit[0] = Color.red;
varit[1] = orange;
varit[2] = Color.yellow;
varit[2] = Color.green;
varit[4] = Color.blue;
```

Kuvio 11. Tiiliseinämän läpinäkyvät tiilet ongelma ja sen aiheuttanut koodinpätkä.

Ongelman ratkaisemisen jälkeen määriteltiin mitä tapahtuu, kun pallo törmää tiileen. Ennen tätä muutosta pallon ja tiilien välinen vuorovaikutus oli mailan kaltainen, eli pallo ei rikkonut tiiliä niihin osuessa. Tätä varten luotiin if-ehtolause, jossa pallo hajotti törmätessään ilman pelaajatunnistetta olevat pelioliot.

Peliprojektin seuraava vaihe oli luoda koodi, joka toistaa ääniraitoja pallon törmätessä peliolioihin tai pelialueen reunoihin. Ääniraitoja toistavan koodin kehitys oli melko yksinkertaista, koska Unity tarjosi kattavan äänilähdedokumentaation. Ääniraitojen toistoa määrittelevien koodinpätkien sijoittelun jälkeen testattiin, että ääniraidat toistuvat niin kuin pitääkin. Pallon osuessa ääntä toistavaan peliolioon Unity antoi huomautuksen: ”*PlayOneShot was called with a null AudioClip*”. Pallon osuessa peliolioon ääniraita toistettiin koodissa asetetulla tavalla, mutta Unitylle ei ollut annettu käytettävää äänitiedostoa. Seuraava oli vuorossa äänitiedostojen lisäys Unity -peliympäristöön ja niiden oikeanlaisen toiston testaus. Äänitiedostojen tallennusmuotona käytettiin *MPEG-1 Audio Layer 3* eli MP3-tallennusmuotoa. Projektissa ei pidetty tärkeänä luoda tehokkaita ääniraitoja, koska tutkielmassa ääniraitojen esittely on melko hankalaa. Tämän takia peliin luotiin muutama helposti toteutettava ääniefekti, joissa napautettiin sormella pöytää.

Viimeiset pääpelinäkymään liittyvät oliot olivat pisteet ja elämien määrä. Näitä olioita varten luotiin uusi piirtoalue ja kaksi käyttöliittymäoliota. Käyttöliittymäoliot olivat tekstiolioita, joihin pystyi kirjoittamaan oletuksena Arial-kirjasintyypillä tekstiä. Pisteet olioon kirjoitettiin ”PISTEET 0” ja elämät olioon kirjoitettiin ” 0 ELÄMÄT”. Nolla tässä tapauksessa esitti paikkamerkkiä, johon koodilla sijoitettiin uutta tietoa. Breakout-pelissä pelaajalla on viisi elämää, mutta elämien määrä tuntui testauksessa olevan liian iso. Tämän seurauksena elämien määrä vähennettiin vain kolmeen. Käyttöliittymän tietojen päivittämistä, pisteiden sekä elämien tarkkailua varten luotiin C#-skriptitiedosto nimeltä PelinOhjain. PallonLiike-tiedostoon lisättiin koodi pallon törmäysten vaikutuksesta pisteiden ja elämien määrään. Pallon osuessa pelialueen alareunaan pelaaja menettää yhden elämän ja pallo annetaan taas pelaajan haltuun. Pallon osuessa tiileen pelaajalle annetaan yksi piste. Piste- ja elämäjärjestelmien toimivuus oli melko samankaltainen, joten elämäjärjestelmän toimivuutta testattiin ennen pistejärjestelmän lopullista toteutusta.



Kuva 10. Kanban-taulu, jossa tehtävänä oli pistejärjestelmän luonti ja elämäjärjestelmän testaus.

Toimivien piste- ja elämäjärjestelmien jälkeen oli vuorossa kehittää ”peli ohi” -näyttö. Uuden näytön kehitys aloitettiin lisäämällä PallonLiike -skriptitiedostoon if-ehtolause, jossa käskettiin pelin siirtyvän uuteen PeliOhi-näkymään pelaajan elämien loppuessa. Pelissä on myös mahdollista hajottaa koko tiiliseinä, joka johtaa uuden tiiliseinän luomiseen. Pallotiedostoon lisättiin tätä mahdollisuutta varten toinen if-ehtolause, jossa käsketään pelin lataavan pelinäkymän uudelleen koko tiiliseinämän hajottua.

```
// Jos tiiliLuku on 25, lataa uusi "Peli" pelinäkymä. 25 on
tiilien yhteissumma.
      if (tiiliLuku == 25)
      {
            PlayerPrefs.SetInt("edellisetpisteet",
            pelinOhjain.Pisteet);
            SceneManager.LoadScene("Peli");
      }
```

PallonLiike –skriptitiedostoon lisättiin myös toiminto, joka tallentaa pelaajan saavuttamat pisteet sekä näyttää ne pelaajalle loppunäkymässä. Uuteen loppunäkymään lisättiin viisi uutta tekstioliota: peliOhi, pisteet, uusiHuippuTulos, paluu ja laskenta. Pisteitä näyttävä olio esitti pelaajan pelissä saavuttaman pisteiden määrää. Huipputulosta osoittava olio ilmestyy pisteiden yläpuolelle, jos pelaaja saavuttaa edellistä tulosta korkeamman pistemäärän. Uusia toimintoja varten luotiin PeliOhiOhjain-skriptitiedosto. Skriptitiedoston tarkoituksena oli tarjota tekstioliolle pelaajan saavuttaman pisteiden tulokset ja aloittaa lähtölaskentatoiminta. Lähtölaskenta alkaa automaattisesti loppunäkymään saavuttaessa. Laskennan loputtua pelaaja siirretään seuraavaksi kehitettävään PaaValikkonäkymään.

Pelin päävalikkoon luotiin kolme uutta tekstioliota: otsikko, huipputulos ja pelaaPelia. Päävalikkonäkymän lisäksi luotiin viimeinen C#-skriptitiedosto nimeltä PaaValikonOhjain. Päävalikkoa varten luotu skriptitiedosto haki pelin loppunäkymästä pelaajan saavuttamat pistetiedot. Pistetiedot näytettiin tämän jälkeen huipputulosoliossa päävalikossa. Päävalikossa oleva pelaaPelia-olio kehottaa pelaajaa painamaan hiiren vasenta näppäintä, jotta pelinäkymä vaihtuu itse pelinäkymään. Peliprojektissa oli tässä vaiheessa luotu toimivat päävalikko, peli ja peli ohi näkymät. Projektin seuraava vaihe oli koota näkymistä yksi toimiva peli ja aloittaa pelin kokonaisuutta tarkasteleva testausvaihe.

5.3.4 Testaus

Peliprojektissa testaus on osa kehitystä ja tämän takia jokaista projektin osaa oli jo testattu yksittäin. Testausvaiheessa oli tarkoitus testata pelin toimivuutta pelaajan

näkökulmasta Unityn kehitysalustan ulkopuolella. Peli käynnistettiin avaamalla pelin ohjelmatiedosto ja heti pelin käynnistettyä ilmeni ongelma. Pelin päävalikossa piti olla hiiren vasemman näppäimen painamista kehottava tekstiolio, mutta tässä tapauksessa kyseinen olio oli kadonnut. Kanban-tauluun lisättiin uusi kadonneeseen tekstiolioon liittyvä ongelmaohjauskortti. Kyseessä oli vain puuttuva tekstiolio, joka ei hakenut uutta tietoa koodista. Ongelman ratkaisu oli tämän takia melko yksinkertainen, koska kyseistä ongelmaa pystyi vertaamaan muihin näkyviin tekstiolioihin. Ongelma johtui tekstille annetun tilan puutteesta. Tekstiolio on kenttä, johon voi lisätä tekstiä tai symboleja. Tässä tapauksessa annettu kenttä oli pienempi kuin käytössä oleva kirjasinkoko, joka johti tekstin katoamiseen. Unityn kehitysympäristössä kyseinen teksti oli testauksessakin näkyvä, mutta pelin siirtyessä omaan ohjelmatiedostoon teksti oli kadonnut. Ongelman korjauksen jälkeen luotiin uusi ohjelmatiedosto ja peli käynnistettiin uudelleen toista testaus kertaa varten.

Toisella testauskerralla ilmentyvä ongelma ei ollut virheellinen koodi tai huonosti toimiva toiminto. Ongelman lähde oli tällä kertaa pelaajan puuttuva kyky lopettaa peli. Kyseessä oli jälleen kerran ongelma, jonka voi havaita vain pelin omassa ohjelmatiedostossa. Kanban-tauluun lisättiin tämän havainnon perusteella uusi tehtäväohjauskortti, jonka aiheena oli luoda pelin lopettava toiminto. Pelin lopettamista varten luotiin yksinkertainen koodinpätkä, joka sammuttaa pelin pelaajan painaessa esc-näppäintä.

```
// ESC-näppäin sulkee pelin
if (Input.GetKey("escape"))
{
    Application.Quit();
}
```

Kolmannella testauskerralla pelissä ei havaittu ongelmia, joten seuraavana projektin vaiheena oli pelin visuaalisen ilmeen parantaminen.

5.3.5 Visuaalisen ilmeen parantaminen

Ensimmäinen visuaalisen ilmeen parantamisen kohde oli pelin kirjasintyypin vaihtaminen. Unity tarjosi oletuksena Arial-kirjasintyypin, joka on selkeä ja helposti luettava kirjasintyyppi. Arial ei kuitenkaan sopinut pelin kirjasintyypiksi, koska Breakout -pelin kirjasintyyppi on huomattavasti kulmikkaampi. Unity tukee *TrueType*- (.ttf) ja *OpenType Fonts* (.otf) -kirjasintyyppejä, joita voi ostaa tai ladata ilmaiseksi internetistä. Peliprojektia varten ladattiin FontSpace-verkkosivustolta ilmainen Chequered Inkin luoma *Premier 2019*-kirjasintyyppi. *Premier 2019*-kirjasintyyppi on hyvin minimalistinen ja kulmikas kirjasintyyppi, joka sopi peliprojektiin hyvin. Kaikki peliprojektin tekstioliot vaihdettiin käyttämään kyseistä uutta kirjasintyyppiä.

Unityn peliympäristön oletus taustaväri on sininen, mutta Breakout -pelissä taustan väri on musta. Taustavärin vaihtaminen on Unityssä helppoa ja se tapahtui painamalla pääkameran taustavärikenttää. Kentän painaminen avasi uuden valikon, jossa väriä voi vaihtaa RGB- tai heksadesimaaliarvoja muuttamalla.

Päävalikossa pelin otsikko oli Arial-kirjasintyypillä kirjoitettu tekstiolio, mutta suunnitelmien mukaan kyseinen otsikko oli vain paikkamerkki. Otsikon tekstiolio poistettiin ja tilalle lisättiin Photoshopilla luotu kuva, joka lisättiin peliympäristöön kuvaoliona. Pelin uudessa otsikkokuvassa luki isoilla kirjaimilla kirjoitettu ”PELI”, joka oli tyylitelty värikkäällä tiilikuvioinnilla. Peliympäristöön lisättiin myös toinen Photoshopilla luotu kuvaolio, joka kuvasi tietokoneen hiirtä. Uusi kuva lisättiin pelikehotuksessa olevien sulkeiden sisälle. Lisätyn hiirikuvan tarkoituksena oli tarkentaa pelikehotusta tarjoamalla pelaajalle visuaalinen avuste.

Kuva 11. Pelin päävalikko.

Pelinäkymässä Arial-kirjasintyypit vaihdettiin päävalikon tapaan. Valkoiset pallo- ja mailaoliot saivat uuden punaisen värin ja pelialueelle lisättiin harmaat reunat. Breakout -pelin Atari 2600 -konsoliversiossa piste- ja elämäoliot ovat reunojen ulkopuolella. Peliprojektin versiossa pisteet ovat pelattavan alueen sisällä, koska pallon liikkuminen pisteiden läpi havaittiin olevan melko hauska ominaisuus.

Kuva 12. Pelin pelinäkymä.

Pelin loppunäkymässä vaihdettiin myös Arial-kirjasintyypit uuteen *Premier 2019*-kirjasintyyppiin. Unity ei tarjoa alleviivausta kirjasintyypeille, vaan käyttäjien täytyy itse ladata Unityyn alleviivauksen omistava kirjasintyyppi. *Premier 2019*-kirjasintyyppiä ei ole alleviivatussa muodossa, joten alleviivaus loppunäkymän otsikkoon luotiin lisäämällä alleviivausta muistuttava kuvaolio.

Kuva 13. Pelin loppunäkymä, jossa on saavutettu uusi huipputulos.

Pisteitä näyttävän tekstiolion yläpuolelle ilmestyi keltaisen värinen huipputuloksesta ilmoittava tekstiolio pelaajan saavuttaessa uuden huipputuloksen. Loppunäkymässä oli myös lähtölaskenta tekstiolio, joka laski viidestä nollaan. Laskurin vaihduttua nollaan peli siirsi pelaajan takaisin päävalikkoon, jossa pelaaja voi aloittaa pelin uudelleen.

6 Tulokset

Tässä luvussa käsitellään tutkielman tapauksessa havaitut keskeisimmät tulokset. Tulokset ovat jaettu kahteen eri osioon: Kanbanin käyttö projektinhallintamenetelmänä ja itsenäisesti toteutettu pelinkehitys. Tulokset ovat jaettu fenomenologista analyysia varten, jotta projektista saatuja havaintoja voidaan pohtia tarkemmin.

6.1 Kanban projektinhallintamenetelmänä

Kanban-taulu on Kanbanin näkyvin ja tunnetuin ominaisuus, joten tämän takia taulun toimivuutta tarkasteltiin tutkielmassa huolellisesti. Kanbanin suuri etu on menetelmän helppo ymmärrettävyys ja yksinkertaisuus. Tutkielmassa toteutetussa pelinkehitysprojektissa nämä ominaisuudet tulivat hyvin esille. Kanban-taulua käytettäessä ei projektissa tarvinnut missään vaiheessa turvautua tarkempiin käyttöohjeisiin. Kanban Tool -palvelun tarjoamat lyhyet selitykset taulun toiminnoista olivat riittävät saattamaan projekti alulle.

Kanban-taulussa olevat ohjauskortit toimivat hyvänä visuaalisena avusteena projektin kulun seuraamiseen. Tehtävien valmistuttua ohjauskorttien siirtäminen tehtyjen tehtävien sarakkeeseen toimi hyvänä motivaation lähteenä. Uusien tehtävien lisääminen tauluun toimi mutkattomasti ja tämä mahdollisti projektin nopean mukautumisen muuttuneeseen tehtävävirtaan. Tehtävävirtaa saattoi myös muuttaa tehtävien tärkeyden vaihtaminen. Ongelmatapauksissa ongelmatehtäville voitiin lisätä korkea tärkeysjärjestys, jolloin uusi ilmestynyt tehtävä voitiin ottaa heti työn alle. Taulussa samanaikaisesti tehtävien työtehtävien ylärajaksi oli asetettu kaksi tehtävää. Rajoituksen tarkoituksena oli mahdollistaa samanaikainen tehtävien testaaminen ja kehittäminen. Rajoituksen yläraja oli alhainen, jotta projektin tärkeimmät työtehtävät saisivat tarvittavan huomion. Tehtävien liiallinen moniajo voi johtaa heikentyvään työn laatuun.

Tutkielman kehitysprojektissa oli käytössä Kanban-tauluohjelmisto, joten kyseessä ei ollut yhdessä paikassa sijaitseva fyysinen taulu. Tämä mahdollisti Kanban-taulun muokkaamisen ja seuraamisen sijainnista huolimatta. Fyysisessä taulussa on myös mahdollisuus kadottaa ohjauskortteja, kun taas virtuaalisessa taulussa mahdollisesti hukkuneet ohjauskortit voi paikantaa tehtävähistoriasta.

Kanban-taulun vahvuus voi samalla olla myös sen heikkous. Tutkielman projekti oli melko lyhyt ja yksinkertainen, mutta tästä huolimatta taulun koko kasvoi huomattavan nopeasti. Kanban-tauluun tehtävien lisäys kasvattaa taulun kokoa ja kasvaneessa tehtäväjonossa voi olla hankala hahmottaa mitä tehtäviä projektissa pitäisi tehdä. Suuresta ohjauskorttien määrästä johtuva sekasorto voi johtaa aikaa vievään hukkaan. Tutkielmassa ohjauskorttien liiallista kasautumista hillittiin luomalla tehtäviä, joihin oli liitetty muistilista. Esimerkiksi pelin visuaalisen ilmeen parantamiseen liittyvä uusien kuvien luominen oli tiivistetty yhteen ohjauskorttiin, jossa erilliset kuvatehtävät olivat muistilistassa.

Taulun toinen mahdollinen ongelma on tehtävien pitäminen ajan tasalla. Taulun ajan tasalla pitämisen laiminlyöminen on taulun koon lisäksi yksi mahdollinen sekasortoa aiheuttava tekijä. Tutkielmassa taulun ajan tasalla pitäminen oli tärkeää projektin dokumentaatiota varten.

6.2 Itsenäisesti toteutettu pelinkehitys

Itsenäisen pelinkehityksen suurin etu on kehittäjän luova vapaus. Pelinkehittäjän ollessa vastuussa pelinkehityksen jokaisesta osa-alueesta mahdollistaa pelin kehityksen kehittäjän oman näkemyksen mukaisesti. Tämä voi olla myös itsenäisen kehityksen huono puoli, koska luova vapaus voi johtaa projektin laajuuden lipsumiseen. Lipsumisen lisäksi vapaus luoda mitä kehittäjä haluaa ei takaa, että kehitetty peli olisi hauska. Tutkielmassa luovaa vapautta hillitsi jo olemassa oleva pelin jäljittely. Kehitetyn pelin päävalikko ja loppunäkymät olivat vapaasti luotavia pelinäkymiä, mutta tässäkin tapauksessa vapautta hillitsi alkuperäisen pelin tyylin jäljittely.

Itsenäisessä pelinkehityksessä tutkielmassa ilmeni kuinka tärkeää voi olla mahdollisuus luoda omat aikataulut tehtävien suorittamiselle. Tehtävillä oli asetettu valmistumista vastaavat päivänmäärät, mutta kellonaika tehtävien valmistumiselle oli joustava. Tehtäviä pystyi tarvittaessa ratkaisemaan työpäivän loputtua ilman, että siitä aiheutui odottavaa hukkaa.

Tutkielman peliprojektissa ilmeni käyttäjävirheestä johtunut tiiliseinämän läpinäkyvyyden ongelma. Kyseinen virhe oli hyvin yksinkertainen kirjoitusvirhe taulukossa, jossa yksi numero oli kirjoitettu väärin. Läpinäkyvyyttä aiheuttavaa ongelmaa yritettiin löytää tiilien sijaintia, kokoa, määrää ja värienperintää sisältävässä koodissa. Ongelman korjaus yrityksissä jopa itse taulukon värejä vaihdeltiin, mutta väärä numero taulussa vältteli katsetta. Ongelma ratkesi lopulta taulukon kokoa ja värejä muuttaessa. Kyseinen ongelma olisi voinut ratketa huomattavasti nopeammin, jos projektissa olisi ollut mukana toinen työntekijä. Toisen ihmisen tarjoama apu koodin tarkastuksessa on etu, jota itsenäisessä pelinkehityksessä ei välttämättä ole aina saatavilla.

Projektissa työtovereiden puute johti tarpeeseen luoda pelin kaikki osa-alueet itse. Pelinkehitystä varten on tarjolla useita ilmaisia tai maksullisia pohjia ja työkaluja, mutta tarjotut pohjat eivät välttämättä sovi projektin tarpeisiin. Tämän takia projektissa kehitettiin visuaalisuus, äänet ja koodi itsenäisesti. Koodaamista kuitenkin nopeutti Unityn tarjoamat valmiit toiminnot.

Itsenäisessä pelinkehitysprojektissa Kanban-taulun tarjoama visuaalisuus helpotti työtehtävien tärkeyksien järjestämistä. Taulussa kasvanut tehtyjen työtehtävien määrä toimi hyvänä motivaation lähteenä, koska projektiin käytetty aika ja vaivannäkö oli näkyvissä taulua käytettäessä. Tutkielman itsenäisesti toteutettu peliprojekti oli yksinkertainen ja Kanbanin käyttö ainoana projektinhallintamenetelmänä oli riittävä.

Taulukossa 1 on esitetty Kanbanin käytöstä löydetyt hyvät ja huonot puolet itsenäisesti toteutetusta ohjelmistokehitysprojektista. Taulukon sisältö on järjestetty aakkosjärjestykseen, koska Kanbanin hyvät ja huonot puolet ovat projektikohtaisia. Tämän takia Kanbanista löydettyihin ominaisuuksiin ei voi asettaa tärkeysjärjestystä.

Taulukko 1. Kanbanin hyvät ja huonot puolet itsenäisessä ohjelmistokehitysprojektissa.

Kanban itsenäisessä ohjelmistokehitysprojektissa	
Hyvät puolet	**Huonot puolet**
Helppo oppia ja yksinkertainen	Taulun ajassa pitämisen laiminlyönti voi aiheuttaa hukkaa
Joustava ja mukautuva menetelmä	Taulu voi muuttua sekavaksi tehtävämäärän kasvaessa
Projektissa voidaan kehittää pieniä valmiita tuotoksia silloin kuin niitä tarvitaan	Taulussa näkyvien työtehtävien määrä voi laskea työmotivaatiota
Taulun ja ohjauskorttien avulla voi asettaa tehtäville tärkeysjärjestyksen	Työtehtävien ja projektin kesto voi olla hankalaa arvioida
Taulun tarjoaman visualisoinnin avulla voi havaita hukkaa aiheuttavia tekijöitä	
Taulun tehtävien tärkeysjärjestys helpottaa keskittymään tärkeisiin tehtäviin	
Taulun työrajoituksilla voi välttää hukkaa aiheuttavaa tehtävien moniajoa	
Taulun uimaratojen avulla voi erottaa projektin eri työluokat toisistaan	
Taulussa näkyvien tehtyjen tehtävien määrä voi kohottaa työmotivaatiota	
Yksinkertaisessa itsenäisesti toteutetussa projektissa Kanban toimii hyvin ainoana projektinhallintamenetelmänä	

7 Diskussio

Tämän tutkielman tavoitteena oli tarkastella Kanbanin soveltuvuutta ohjelmistokehitykseen tuottamalla pelinkehitysprojekti ja vertaamalla projektista saatuja tuloksia aikaisempiin tutkimuksiin. Tutkielman aineiston ja projektin analysointia varten käytettiin Fenomenologista analyysia. Fenomenologinen analyysi avulla tutkimuksen tuloksia verrattiin jo olemassa oleviin tutkimuksiin Kanbanin soveltuvuudesta ohjelmistokehityksen ketterään projektinhallintaan. Tapaustutkimuksen tutkimuskysymyksenä oli:

- *Kuinka Kanban soveltuu ketteränä projektinhallintamenetelmänä ohjelmistokehitykseen?*

Tutkielman projektin tapauksena oli itsenäinen pelinkehitysprojekti, jota ohjattiin käyttämällä Kanbanin tarjoamaa Kanban-taulua. Peliprojektin kohteena oli kehittää Atari Breakout -peliä jäljittelevä ohjelma. Projektissa noudatettiin Kanbanin jatkuvaa työvirtaa, eli mahdolliset uudet tehtävät voitiin aloittaa heti niiden ilmestyttyä. Projektin alkupuolella Kanbanin mukautuvuutta testattiin lisäämällä Kanban-tauluun tehtävä, jossa aiheena oli odottamattoman ongelman korjaaminen. Ongelma saatiin korjattua onnistuneesti ilman, että projektissa koettiin merkittäviä katkoksia.

Tutkielman tutkimustavoitteet saavutettiin suunnitelmien mukaan. Tutkielman tuloksena oli toimiva tietokonepeli, jonka kehitysprosessista saatuja tuloksia voitiin käyttää tutkimuksen havaintojen analysoinnissa. Havaintojen analysoinnin pohjalta voitiin muodostaa vertauskohteita jo olemassa oleviin tutkimuksiin Kanbanista ohjelmistokehityksessä. Tämän lisäksi voitiin tarkastella, kuinka saadut tulokset vaikuttivat käytäntöön sekä mahdollisiin jatkotutkimuksiin.

7.1 Tulosten vertailu aikaisempiin tutkimuksiin

Kanban on ketterä projektinhallintamenetelmä. Tämän takia Kanbanin käyttö ja siitä saadut tulokset ovat projekti- ja ryhmäkohtaisia. Kanbania voidaan myös käyttää osana toista projektinhallintamenetelmää muodostaen hybridiprojektinhallintamenetelmän. Scrumban on yksi Kanbania käyttävistä projektinhallintamenetelmistä, jossa Kanbania käytetään projektin visualisoinnissa ja ohjaamisessa Kanban-taulun avulla.

Tutkielman tapaus oli itsenäisesti toteutettu pelinkehitysprojekti. Tämän takia tutkimuksen tulokset voivat erota tutkimuksista, jossa Kanbania on käytetty suurien projektien projektinhallintamenetelmänä. Kanbanista on kuitenkin tehty hyvin vähän tutkimusta verrattuna muihin ketteriin projektinhallintamenetelmiin, kuten esimerkiksi Scrum. Ahmadin, Markkulan ja Oivon (2013) tuottamassa kirjallisuuskatsauksessa selvisi, että Kanbanista oli tehty vain yhdeksäntoista tutkimusta vuodesta 2000 vuoteen 2011. Digital.ain (2020) julkaisemassa *14th Annual State of* Agile -raportissa Scrum oli projektinhallinnassa käytössä 58 % yrityksistä, kun taas Kanbanin käyttö oli vain 7 %. Tutkimusten ja käytön määrän niukkuuden takia tutkielmasta saadut itsenäisen projektin tulokset olivat hyvä lisä Kanbanin käytännön kannalta.

Ikonen ja muut (2011) tapaustutkimuksessa selvisi, että Kanbanin arvo projektinhallinnassa muodostuu Kanbanin kyvystä visualisoida projektin kulku ja tehtävät reaaliajassa. Tutkielman peliprojektissa käytössä olleesta Kanban-taulusta saadut tulokset tukevat tätä väitettä. Kanban-taulusta saatu palaute auttoi ohjaamaan projektin kulkua. Kanban-taulussa tehtyjen tehtävien visualisointi motivoi jatkamaan kehitystä, koska projektiin käytetty vaivannäkö oli visuaalisesti nähtävillä.

Ikonen ja muut (2011) tapaustutkimuksessa todettiin, että Kanban on hyvin yksinkertainen hallintatyökalu ja tämän takia Kanban ei yksinään riitä takaamaan onnistunutta projektia. Scrumban on yksi käytetyimmistä menetelmistä juuri siksi, että se tarjoaa Scrumista tutun kehitysprosessin Kanbanin avulla visualisoituna. Tässä tutkielmassa Kanbanin käyttö yksinään projektinhallintamenetelmänä ei aiheuttanut ongelmia, koska

projekti oli itsenäisesti toteutettu ja yksinkertainen. Tuloksia Kanbanin käytöstä itsenäiseen ohjelmistokehitykseen tukee myös Chris Estevezin (2015) artikkeli, jossa hän omien kokemuksiensa perusteella piti Kanbania tehokkaana valintana itsenäisessä työskentelyssä.

Kanbanin toimivuutta ohjelmistokehityksessä tutkittiin tutkielmassa tuottamalla tietokonepeliprojekti. *Game Development and Production* -käsikirjassa Erik Bethke toteaa, että pelinkehitys on osa ohjelmistokehitystä ja molemmat kehitysalueet jakavat samankaltaiset kehitysprosessit (Bethke, 2003). Murphy-Hill ja muut (2014) tuottamassa tutkimuksessa kuitenkin ilmeni, että kehitysprosessit voivat kuitenkin erota toisistaan melko paljon. Tämän takia kaikkia ohjelmistokehityksen prosesseja ei voi välttämättä käyttää tehokkaasti pelinkehityksessä. Huomattavin eroa aiheuttava tekijä oli pelinkehityksen subjektiivisuus ja hauskuuden tavoittelu. Tästä huolimatta pelinkehitys jakoi ohjelmistokehityksen kanssa samat projektinhallintamenetelmät. Tämä johtui ketterien menetelmien kyvystä mukautua projektin tarpeisiin. Tutkielman tapauksessa Kanban mukautui pelinkehityksen prosesseihin tarpeen mukaan.

7.2 Tulosten merkitys teorialle

Tutkielmassa esille tulleet löydökset tukivat jo olemassa olevaa teoriaa Kanbanin käytöstä ohjelmistokehityksessä. Huomattavin poikkeus jo olemassa olevien tutkimuksien tuloksiin tuli Kanbanin soveltuvuudesta itsenäiseen kehitystyöprojektiin. Tutkielman tuloksissa Kanbanin toiminta itsenäisessä kehitystyöprojektissa oli riittoisa. Tämä kuitenkin poikkesi Ikonen ja muut (2011) tapaustutkimuksen tuloksista, jossa todettiin Kanbanin tarvitsevan tukea lisäkäytännöillä.

Itsenäisestä pelinkehityksestä on löytyvillä hyvin niukasti tutkimusta ja usein tietoa löytää vain artikkeleista kokemuskirjoituksissa. Tästä johtuen itsenäiset ja pienryhmä kehittäjät voivat sivuuttaa Kanbanin projektinhallintamenetelmänä. Tämä tutkielma pyrki tuottamaan lisätuloksia Kanbanin käytöstä itsenäisessä pelinkehityksessä, jotta

projektinhallintamenetelmän valinta kehitystyöhön olisi helpompaa. Projektit voivat olla hyvinkin yksilökohtaisia, joten projektinhallintamenetelmän valintaan ei ole yhtä oikeaa vastausta. Tämän takia projektinhallintamenetelmistä täytyy tehdä mahdollisimman paljon erilaista tutkimusta, jotta menetelmän valinta olisi helpompaa.

7.3 Tulosten merkitys käytännölle

Videopeliala on kasvanut nopeasti ohjelmistokehityksen rinnalla ja nopean kasvun mukana seurasi erilaiset ohjelmisto- ja pelinkehityksen työkalut ja oppaat. Uudet työkalut ja oppaat ovat antaneet käyttäjille mahdollisuuden luoda omia pelejä taustatiedoista huolimatta. Tämän lisäksi digitaaliset latauspalvelut ovat mahdollistaneet pelien julkaisemisen ilman tarvetta fyysisille pelikopioille. Fyysisistä kopioista säästetyt varat voidaan käyttää pelin lisäkehitykselle tai mainostamiselle. Pelialalle pääsyn kynnyksen alentuminen on johtanut useiden erilaisten indie-pelien julkaisemiseen. Osa tuotetuista indie-peleistä oli alun perin yhden ihmisen harrastus tai intohimoprojekti. Esimerkiksi Minecraft sai alkunsa Markus Perssonin luomana itsenäisenä peliprojektina.

Kehitystyö on vain yksi osa peliprojektia ja projektin laajuuden kasvaessa kehitystyö voi hankaloitua ilman projektinhallintatyökaluja. Scrum on yksi suosituimmista ketteristä projektinhallintamenetelmistä ohjelmisto- ja pelinkehitykseen. Scrum ei ole välttämättä paras valinta itsenäiseen tai pienen ryhmän tuottamaan projektiin. Scrumista tutut päivittäiset kokoukset voivat myös olla ongelma, koska ajatus päivittäisistä kokouksista saattaa pelottaa kehittäjiä. Kanbanin yksinkertaisuuden ja visualisoinnin vuoksi Kanban saattaa soveltua Scrumia paremmin itsenäiseen ja pienryhmien pelinkehitysprojekteihin.

Tämän tutkielman tarkoitus oli tuottaa lisää tietoa Kanbanin soveltuvuudesta pelinkehitykseen, varsinkin itsenäisen ja pienryhmien kannalta. Sopivalla projektinhallintamenetelmän valinnalla voidaan säästyä turhalta projektin hukalta. Kanbanista niukasti löytyvän tutkimuksen määrä voi johtaa menetelmän sivuuttamiseen mahdollisena työkaluna projektinhallintaan. Tutkielman tuloksissa vahvistettiin Kanbanin soveltuvuutta

projektinhallinnan visualisoinnin työkaluna. Tämän takia Kanban voi olla projektissa hyvä tukea antava menetelmä, kuten esimerkiksi Scrumban hybridimenetelmässä.

7.4 Tulosten merkitys jatkotutkimuksille

Ohjelmisto- ja pelinkehitykselle ei ole olemassa yhtä oikeaa valintaa projektinhallintamenetelmäksi. Valintaan voi vaikuttaa projektin koko, monimutkaisuus, laajuus tai ryhmän kokemus projektityöskentelystä. Jokainen ohjelmisto- ja pelinkehitysprojekti on omanlaisensa ja tämän takia hallintamenetelmän valinta ei välttämättä ole yksinkertainen ongelma. Tästä johtuen ketteristä projektinhallintamenetelmistä on tehtävä mahdollisimman paljon erilaista tapaustutkimusta, jotta hallintamenetelmän valinta olisi mahdollisimman helppoa.

Tässä tutkielmassa tutkittiin Kanbanin käyttöä itsenäisessä kehitystyöprojektissa, joten tapauksesta saadut tulokset eivät välttämättä vastaa suuren ryhmän tuottaman projektin tuloksia. Tutkielman tulokset toimivat hyvänä vertauskohteena Kanbanin skaalautuvuudesta. Jatkotutkimuksena Kanbanin toimivuutta voitaisiin testata suuremmassa projektissa ja sitä, mitä ongelmia Kanbanin käyttö ainoana menetelmänä suuressa projektissa aiheuttaa.

Tutkielmassa tarkasteltiin Scrumin toimintaa ja hybridimenetelmiä. Jatkotutkimuksen kannalta Scrumbanista voitaisiin tehdä tapaustutkimus, jossa testataan Kanbanin ja Scrumin ominaisuuksia samalla. Hybridimenetelmät voivat toimia hyvänä välimenetelmänä projektinhallintamenetelmää vaihdettaessa. Hybridimenetelmillä voidaan opettaa projektin jäsenille uusia käytäntöjä ilman, että projekti jouduttaisiin keskeyttämään. Jatkotutkimus voitaisiin toteuttaa Scrumia käyttävän ryhmän sopeutumisesta Scrumbanin käyttöön.

Lähteet

Adobe Inc. (2021a). ‘*General trademark guidelines’* Noudettu 4. Tammikuuta osoitteesta: https://www.adobe.com/fi/legal/permissions/trademarks.html

Adobe Inc. (2021b). ‘*File formats’* Noudettu 4. Tammikuuta 2021 osoitteesta: https://helpx.adobe.com/photoshop/using/file-formats.html#photoshop_format_psd

Adobe. (2021). ‘*Tee. Usko. Photoshop.’* Noudettu 4. Tammikuuta 2021 osoitteesta: https://www.adobe.com/fi/products/photoshop.html

Agile Manifesto. (2001). *’Julistuksen takana olevat periaatteet’* Noudettu 14. Joulukuuta 2020 osoitteesta: https://agilemanifesto.org/iso/fi/principles.html

Ahmad, M., O., Markkula, J. ja Oivo, M. (2013). *‘Kanban in software development: A systematic literature review’* 39th Euromicro Conference on Software Engineering and Advanced Applications, s. 9-16. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1109/SEAA.2013.28

Airbrake. (2016). ‘*Waterfall Model: What Is It and When Should You Use It?*’ Noudettu 9. Tammikuuta 2021 osoitteesta: https://airbrake.io/blog/sdlc/waterfall-model

Bethencourt, M. (2016). ‘*Solo Game Dev Kanban Magic: I used to struggle with self-management, but my most recent technique seems to be working really well’* Noudettu 3. Helmikuuta 2021 osoitteesta: https://michaelb.org/solo-game-dev-kanban-magic/

Bethke, E. (2003). *‘Game Development and Production (Wordware Game Developer's Library)’* Plano, Texas: Wordware Publishing, Inc. ISBN 978-1556229510

Boiser, L. (2020). ‘*Swarming as a Kanban Team’* Noudettu 26. Joulukuuta 2020 osoitteesta: https://kanbanzone.com/2020/swarming-as-a-kanban-team/

Caruso, N. (2018). *'The Story of Spyro the Dragon | Gaming Historian'* Noudettu 10. Tammikuuta 2021 osoitteesta: https://youtu.be/faV6qLqBAPc

CAST. (2014). '*New Report Confirms Agile/Waterfall Mix Produces the Best Code Quality'* Noudettu 26. Joulukuuta 2020 osoitteesta: https://www.castsoftware.com/discover-cast/press-releases/docs/press-releases/new-report-confirms-agile-waterfall-mix-produces-the-best-code-quality

Chee-Hong, H. (2018). '*Common Misinterpretations of Scrum'* Noudettu 20. Joulukuuta 2020 osoitteesta: https://www.scrum.org/resources/blog/common-misinterpretations-scrum

Chiu, Y. (2010). *'An Introduction to the History of Project Management: From the Earliest Times to A.D.1900'* Delft: Eburon, ISBN 978-90-5972-437-2

CPM Scheduling. (2019). '*CPM Advantages and Disadvantages'* Noudettu 11. joulukuuta 2020 osoitteesta: https://www.cpmscheduling.com/critical-path-method/advantages-and-disadvantages-in-the-implementation-of-cpm/

De Vynck, G. (2020). '*Unity Technologies Aims to Bring Video Game Tools into the Real World'* Noudettu 18. Tammikuuta 2021 osoitteesta: https://www.bloomberg.com/news/articles/2020-05-07/unity-technologies-aims-to-bring-video-game-tools-into-the-real-world

Dealessandri, M. (2020). '*What is the best game engine: is Unity right for you?'* Noudettu 18. Tammikuuta 2021 osoitteesta: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

Dictionary. (2020). '*Kanban*' Noudettu 22. Joulukuuta 2020 osoitteesta: https://www.dictionary.com/browse/kanban

Digital.ai. (2020). *'14th Annual State of Agile Report'* Noudettu 12. Tammikuuta 2021 osoitteesta: https://stateofagile.com/

Dove, J. (2013). '*Adobe scraps Creative Suite software licenses in favor of cloud subscriptions*' Noudettu 4. Tammikuuta 2021 osoitteesta: https://www.macworld.com/article/2037034/adobe-scraps-software-licenses-in-favor-of-cloud-subscription-scheme-for-creative-suite-line.html

Eby, K. (2018). *'Demystifying the 5 Phases of Project Management'* Noudettu 6. Joulukuuta 2020 osoitteesta: https://www.smartsheet.com/blog/demystifying-5-phases-project-management

Epic Games. (2021). 'Frequently Asked Questions (FAQ)' Noudettu 10. Tammikuuta 2021 osoitteesta: https://www.unrealengine.com/en-US/faq

Estevez, C. (2015). '*Project Planning for Solo Game Developers'* Noudettu 3. Helmikuuta 2021 osoitteesta: http://hacknplan.com/project-planning-for-solo-game-developers/

Fridman, A. (2016). '*The Massive Downside of Agile Software Development*' Noudettu 30. Joulukuuta 2020 osoitteesta: https://www.inc.com/adam-fridman/the-massive-downside-of-agile-software-development.html

Granulo, A., ja Tanović, A. (2019). 'Comparison of SCRUM and KANBAN in the Learning Management System implementation process' 27th Telecommunications Forum (TELFOR), s. 1-4. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1109/TELFOR48224.2019.8971201

Haughey, D. (2014). '*A Brief History of Project Management'* Noudettu 4. Joulukuuta 2020 osoitteesta: https://www.projectsmart.co.uk/brief-history-of-project-management.php

Hello Games. (2016). *'Update 1.03'* Noudettu 10. Tammikuuta 2021 osoitteesta: https://www.nomanssky.com/2016/08/update-1-03/?

Highsmith, J. (2001). '*History: The Agile Manifesto'* Noudettu 14. Joulukuuta 2020 osoitteesta: https://agilemanifesto.org/history.html

Hirschtick, J. (2020). '*The Road to SaaS: The Current Trends Driving Adoption in Product Development*' Noudettu 30. Joulukuuta 2020 osoitteesta: https://www.ptc.com/en/blogs/corporate/2020-product-development-roads-lead-to-saas

Hirsjärvi, S., Remes, P, ja Sajavaara, P. (2009). *'Tutki ja kirjoita'* Helsinki, Suomi: Tammi ISBN 9789513148362

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P. ja Abrahamsson, P. (2011). *'On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation'* 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, NV, USA, 2011, s. 305-314. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1109/ICECCS.2011.37

Ikonen, M., Kettunen, P., Oza, N., ja Abrahamsson, P. (2010). *'Exploring the Sources of Waste in Kanban Software Development Projects'* 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, Lille, France, 2010, s. 376-381. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1109/SEAA.2010.40

JetBrains. (2021). '*Rider - Fast & powerful, cross-platform .NET IDE'* Noudettu 18. Tammikuuta 2021 osoitteesta: https://www.jetbrains.com/rider/

Jyväskylän yliopisto. (2015a). *'Tapaustutkimus'* Noudettu 6. Helmikuuta 2021 osoitteesta: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/tapaustutkimus

Jyväskylän yliopisto. (2015b). *'Fenomenologinen analyysi'* Noudettu 6. Helmikuuta 2021 osoitteesta: https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/aineiston-analyysimenetelmat/fenomenologinen-analyysi

Kanban Tool. (2020). '*A History of Kanban*' Noudettu 22. Joulukuuta 2020 osoitteesta: https://kanbantool.com/kanban-guide/kanban-history

Kanban Tool. (2021a). *'Pricing'* Noudettu 16. Tammikuuta 2021 osoitteesta: https://kanbantool.com/pricing

Kanban Tool. (2021b). *'Kanban Tool On-Site'* Noudettu 16. Tammikuuta 2021 osoitteesta: https://kanbantool.com/kanban-tool-on-site

Kanbanize. (2020a). *'What Does Agile Mean in Project Management?'* Noudettu 14. Joulukuuta 2020 osoitteesta: https://kanbanize.com/agile/project-management

Kanbanize. (2020b). '*What Is a Kanban Card? Details and Example'* Noudettu 22. Joulukuuta 2020 osoitteesta: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban-card

Kastrenakes, J. (2020). '*How photoshop became a verb'* Noudettu 4. Tammikuuta 2021 osoitteesta: https://www.theverge.com/2020/2/19/21143794/photoshop-30th-anniversary-adobe-verb-origin-story

Kasurinen, J., Palacin-Silva, M., Vanhala, E. (2017). *'What Concerns Game Developers? A Study on Game Development Processes, Sustainability and Metrics'* 2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM), s. 15-21. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1109/WETSoM.2017.3

Khan, A. (2019). *'In-house game engines'* Noudettu 10. Tammikuuta 2021 osoitteesta: https://medium.com/@azamkhandev/in-house-game-engines-acca023a1c70

Koutonen J., Leppänen M. (2013). *'How Are Agile Methods and Practices Deployed in Video Game Development? A Survey into Finnish Game Studios.'* Baumeister H., Weber B. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2013. Lecture Notes in Business Information Processing, vol 149. Springer, Berlin, Heidelberg. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1007/978-3-642-38314-4_10

Kukhnavets, P. (2019). *'How Critical Path Method Drives Project Management Success'* Noudettu 10. Joulukuuta 2020 osoitteesta: https://blog.ganttpro.com/en/critical-path-method-cpm-in-project-management/

Levy, F. K., Thompson, G. L., Wiest, J. D. (1963). *'The ABCs of the Critical Path Method'* Noudettu 10. Joulukuuta 2020 osoitteesta: https://hbr.org/1963/09/the-abcs-of-the-critical-path-method

Lindblad, M. (2020). '*The History of Photoshop – Photoshop Through the Years*' Noudettu 4. Tammikuuta 2021 osoitteesta: https://filtergrade.com/history-of-photoshop-through-the-years/

Logistiikan Maailma. (2020). '*Pull Control and JIT'* Noudettu 22. Joulukuuta 2020 osoitteesta: https://www.logistiikanmaailma.fi/en/logistics/production/pull-control-jit/

Lucidchart Content Team. (2020). '*The 4 phases of the Project management Life Cycle*' Noudettu 7. Joulukuuta 2020 osoitteesta: https://www.lucidchart.com/blog/the-4-phases-of-the-project-management-life-cycle

Microsoft. (2021a). '*Microsoft Software License Terms – Microsoft Visual Studio Community 2019*' Noudettu 19. Tammikuuta 2021 osoitteesta: https://visualstudio.microsoft.com/license-terms/mlt031819/

Microsoft. (2021b). '*Visual Studio IntelliCode – AI-assisted development*' Noudettu 19. Tammikuuta 2021 osoitteesta: https://visualstudio.microsoft.com/services/intellicode/

Moujib, A. (2007). '*Lean Project Management*' Noudettu 23. Joulukuuta 2020 osoitteesta: https://www.pmi.org/learning/library/lean-project-management-7364

Murphy-Hill, E., Zimmermann, T., ja Nagappan, N. (2014). *'Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?'* Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, s. 1–11. Noudettu 10. Tammikuuta 2021 osoitteesta: https://doi.org/10.1145/2568225.2568226

Nayab, N. (2011). '*What Is the Criticism of Lean Manufacturing?*' Noudettu 23. Joulukuuta 2020 osoitteesta: https://www.brighthubpm.com/methods-strategies/105933-criticism-of-lean-manufacturing/

Oberon Gaming. (2019). *'Breakout (1978) [Atari 2600]'* Noudettu 16. Tammikuuta 2021 osoitteesta: https://www.youtube.com/watch?v=Cr6z3AyhRr8

Paananen, P. (2012). *'Photoshop CS6 - kuvankäsittely'* Jyväskylä: Docendo, ISBN 978-952-5912-27-2

Pahuja S. (2016). '*What is Scrumban?*' Noudettu 26. Joulukuuta 2020 osoitteesta: https://www.agilealliance.org/what-is-scrumban/

Pascarella, L., Palomba, F., Di Penta, M., ja Bachelli, A. (2018). *"How Is Video Game Development Different from Software Development in Open Source?*" Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18). Association for Computing Machinery, New York, NY, USA, s. 392–402. Noudettu 10. Tammikuuta 2021 osoitteesta: https://doi.org/10.1145/3196398.3196418

Poole, S. (2020). '*The renaissance and evolution of indie games'* Noudettu 3. Helmikuuta 2021 osoitteesta: https://venturebeat.com/2020/12/16/the-renaissance-and-evolution-of-indie-games/

Project Management Institute. (2017). *'PMBOK® Guide – Sixth Edition*' Project Management Institute, ISBN 978-1-62825-184-5

Project Management Institute. (2019). *'Pulse of the Profession 2019'* Noudettu 26. Joulukuuta 2020 osoitteesta: https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2019

Project Management Institute. (2020). '*What is Project Management?*' Noudettu 6. Joulukuuta 2020 osoitteesta: https://www.pmi.org/about/learn-about-pmi/what-is-project-management

Ramée, J. (2018). '*The 14 Best Games Developed by Only One Person'* Noudettu 3. Helmikuuta 2021 osoitteesta: https://www.gamespot.com/gallery/the-14-best-games-developed-by-only-one-person/2900-2172/

Rehkopf M. (2019). '*What is Kanban Board?*' Noudettu 22. Joulukuuta 2020 osoitteesta: https://www.atlassian.com/agile/kanban/boards

Royce, W. (1970). *'Managing the Development of Large Software Systems'* Noudettu 9. Joulukuuta 2020 osoitteesta: http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

Schwaber, K. (1997). '*SCRUM Development Process'* London: Springer, Noudettu 20. Joulukuuta 2020 osoitteesta: https://doi.org/10.1007/978-1-4471-0947-1_11

Schwaber, K., & Sutherland, J. (2020). '*The 2020 Scrum Guide™'* Noudettu 20. Joulukuuta 2020 osoitteesta: https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Schweda, A., Winkler, D., Biffl, S., & Musil, J. (2010*). 'A Survey on the State of the Practice in Video Game Software Development.'* Institute of Software Technology and Interactive Systems Favoritenstr. s. 9-11, A-1040 Vienna, Austria. Noudettu 12. Tammikuuta 2021 osoitteesta: https://www.researchgate.net/publication/228980033_A_Survey_on_the_State_of_the_Practice_in_Video_Game_Software_Development

Scrum.org. (2019). '*What is Scrum?*' Noudettu 18. Joulukuuta 2020 osoitteesta: https://www.scrum.org/resources/what-is-scrum

Scrum.org. (2020). '*The Scrum Framework Poster'* Noudettu 21. Joulukuuta 2020 osoitteesta: https://www.scrum.org/resources/scrum-framework-poster

Seymour, T. & Hussein, S. (2014). '*The History of Project Management'* International Journal of Management & Information Systems (IJMIS). Noudettu 4. Joulukuuta 2020 osoitteesta: https://www.researchgate.net/publication/298341808_The_History_Of_Project_Management

Sjøberg, D. (2018). *'An empirical study of WIP in kanban teams.'* Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18). Association for Computing Machinery, New York, NY, USA, Article 13, s. 1–8. Noudettu 12. Tammikuuta 2021 osoitteesta: https://doi.org/10.1145/3239235.3239238

Skhmot, N. (2017). '*The 8 Wastes of Lean'* Noudettu 23. Joulukuuta 2020 osoitteesta: https://theleanway.net/The-8-Wastes-of-Lean

Smartsheet. (2017). '*What's the Difference? Agile vs Scrum vs Waterfall vs Kanban'* Noudettu 22. Joulukuuta 2020 osoitteesta: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

Smartsheet. (2020). '*Scrumban: Choosing the Middle Ground Between Scrum and Kanban*' Noudettu 26. Joulukuuta 2020 osoitteesta: https://www.smartsheet.com/scrum-ban-choosing-middle-ground-between-scrum-and-kanban

Smith, J. (2020). '*What is Photoshop'* Noudettu 4. Tammikuuta 2021 osoitteesta: https://www.agitraining.com/design-news/photoshop-training-news/what-photoshop

Takeuchi, H. & Nonaka, I. (1986). *'The New New Product Development Game'* Noudettu 20. Joulukuuta 2020 osoitteesta: https://hbr.org/1986/01/the-new-new-product-development-game

Tassi, P. (2019). '*Todd Howard's 'Fallout 76' Admission Is A Weird Moment for The Industry'* Noudettu 30. Joulukuuta 2020 osoitteesta: https://www.forbes.com/sites/paultassi/2019/06/04/todd-howards-fallout-76-admission-is-a-weird-moment-for-the-industry/

Unity Technologies. (2021a). *'Plans and pricing?'* Noudettu 18. Tammikuuta 2021 osoitteesta: https://store.unity.com/#plans-individual

Unity Technologies. (2021b). *'Bolt Visual Scripting'* Noudettu 18. Tammikuuta 2021 osoitteesta: https://unity.com/products/unity-visual-scripting

Unity Technologies. (2021c). '*Assembly definitions*' Noudettu 24. Helmikuuta 2021 osoitteesta: https://docs.unity3d.com/Manual/ScriptCompilationAssemblyDefinitionFiles.html

University of Pennsylvania. (2020). *'John W. Mauchly Papers'* Noudettu 11. Joulukuuta 2020 osoitteesta: http://dla.library.upenn.edu/dla/ead/detail.html?id=EAD_upenn_rbml_PUSpMsColl925

Verheyen, G. (2020). '*Scrum: A Brief History of a Long-Lived Hype'* Noudettu 20. Joulukuuta 2020 osoitteesta: https://guntherverheyen.com/wp-content/uploads/2020/12/Scrum-A-Brief-History-of-a-Long-Lived-Hype-Paper.pdf

Ward, J. (2008). *'What is a Game Engine?*' Noudettu 10. Tammikuuta 2021 osoitteesta: https://www.gamecareerguide.com/features/529/what_is_a_game_.php

Weaver, P. (2006). *'A Brief History of Scheduling'* Noudettu 10. Joulukuuta 2020 osoitteesta: https://web.archive.org/web/20150518071346/http://www.mosaicprojects.com.au/PDF_Papers/P042_History%20of%20Scheduing.pdf

Wrike. (2020). *'What Is Waterfall Project Management?'* Noudettu 7. Joulukuuta 2020 osoitteesta: https://www.wrike.com/project-management-guide/faq/what-is-waterfall-project-management/