VAATIMUSMÄÄRITTELYN ROOLI KETTERÄSSÄ OHJELMISTOKEHITYKSESSÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
syksy, 2021

Sanna Lilja

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Sanna Lilja Vuosi 2021

Työn nimi Vaatimusmäärittelyn rooli ketterässä ohjelmistokehityksessä

Ohjaajat Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli tarkastella, mitä vaatimusmäärittely on, vaatimusmäärittelyn roolia ketterässä ohjelmistokehityksessä sekä tutkia soveltuuko ketterän ohjelmistokehityksen tapa tehdä vaatimusmäärittelyä pieneen projektiin.

Opinnäytetyön tietopohjassa tarkastellaan ohjelmistoprojektimallien historiaa aina synnystä ketteröitymiseen asti, mitä vaatimusmäärittely on sekä sitä, miten vaatimusmäärittely toteutetaan ketterässä ohjelmistokehityksessä. Opinnäytetyö on toiminnallinen ja sen yhteydessä toteutettiin kehitysprojekti, jossa hyödynnettiin tietopohjassa läpikäytyjä kokonaisuuksia.

Opinnäytetyön tietopohjan ja käytännön työn perusteella voidaan todeta, että vaatimusmäärittelyllä on tärkeä rooli onnistuneessa ohjelmistokehitysprojektissa, projektin koosta ja projektimallista riippumatta. Nopeasti muuttuva maailma haastaa projekteja ja varsinkin vaatimusmäärittelyä. Ketterä ohjelmistokehitys pyrkii vastaamaan tähän haasteeseen muuttamalla tapaa, jolla vaatimusmäärittelyä tehdään. Ketterässä ohjelmistokehityksessä vaatimusmäärittely muuttuu joustavaksi ja sen painopiste siirtyy projektin alusta projektin aikana tapahtuviin iteraatioihin. Näin varmistetaan, että muutoksiin ja muuttuviin vaatimuksiin pystytään reagoimaan. Myös asiakaspalaute nousee tärkeään rooliin ketterän ohjelmistokehityksen tavassa tehdä vaatimusmäärittelyä.

Avainsanat Ohjelmistoprojektimalli, Vaatimusmäärittely, Ketterä ohjelmistokehitys

Sivut 38 sivua ja liitteitä 4 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology **Abstract**

Hämeenlinna University Centre

Author Sanna Lilja Year 2021

Subject The role of defining requirements in Agile software development

Supervisors Lasse Seppänen

ABSTRACT

The purpose of the thesis was to elaborate what requirements are and what role requirements play in Agile software development. Purpose was also to study will Agile way of defining requirement work in a small software development project.

The theoretical part of the thesis observes history of software development models from beginning up to Agile software development model, what defining requirements means and how requirements are defined in Agile software development. The thesis is functional, and the practical work is based on theoretical part of the theses.

The conclusion is that defining requirements has a key role in the success of the software development project, no matter the size or the project model. Turbulent change in the environment is very challenging to the projects and defining requirements. Agile software development is attempting to response to this challenge by changing the way how defining requirements are defined. In Agile software development defining requirements become flexible. Traditionally requirements are defined in the beginning of the project, however in Agile software development, requirements are defined during the project iteration. This enables reaction to the change in requirements. The role of customer feedback become important in the way of defining requirements in Agile software development.

Keywords Software development model, Requirement definition, Agile software development

Pages 38 pages and appendices 4 pages

Sanasto

Asiakas	palveluja käyttävä tai muuten niiden kohteena oleva taho tai tuotteiden vastaanottaja
Asiakkaan edustaja	taho, joka edustaa asiakasta esimerkiksi ohjelmiston tilaaja
Iteraatio	ennalta määritelty ajanjakso, jonka aikana toteutetaan sovittuja työkokonaisuuksia (engl. Sprint) kts. iteratiivinen
Iteratiivinen	syklimäinen tapa kehittää ohjelmistoa. Syklit eli iteraatiot toistuvat niin monta kertaa, kunnes tietty lopputulos saavutetaan kts. Iteraatio.
Kehitysjono	lista ohjelmiston tunnistettuja ominaisuuksista, jotka tulee tehdä, jotta ohjelmisto valmistuu (engl. Backlog)
Ketterä ohjelmistokehitys	käsite, jonka alle sijoittuu ketteriä projektimalleja ja käytäntöjä (engl. Agile software development)
Ketterä projektimalli	projektimalli, joka noudattaa ketterän ohjelmistokehityksen arvoja ja periaatteita esimerkiksi Scrum – menetelmä kts. Ohjelmistoprojektimalli
Käyttäjätarina	vakiomuotoinen lause, joka kuvaa järjestelmän toivotun toiminnallisuuden käyttäjän näkökulmasta ja ohjaa teknistä toteutusta (engl. User story).
Ohjelmistoprojektimalli	projektimalli, joka määrittelee miten ohjelmistoprojektia ja sen kokonaisuuksia johdetaan. Kuvaa projektin elinkaaren vaiheet ja määrittelee projektin roolit.
Projekti	suunniteltu työkokonaisuus tietyn päämäärän saavuttamiseksi

Sisällys

1 Johdanto 1
2 Ohjelmistoprojektimallien synty ja ketteröityminen 2
2.1 Vesiputousmallista iteratiivisten projektimallien yleistymiseen 3
2.2 Ketterän ohjelmistokehityksen synty 6
2.3 Ketterä ohjelmistokehitys 7
2.4 Ketterän ohjelmistokehityksen projektimallit 9
3 Vaatimusmäärittely 11
3.1 Toiminnalliset ja ei-toiminnalliset vaatimukset 11
3.2 Vaatimusmäärittelyn vaiheet 13
3.2.1 Vaatimusten kartoittaminen ja analysointi 14
3.2.2 Vaatimusten dokumentointi ja validointi 15
4 Vaatimusmäärittely ketterässä ohjelmistokehityksessä 16
4.1 Käyttäjätarina 17
4.2 Dokumentaatio 18
4.3 Kehitysjono 19
4.4 Iteraatio vaatimusmäärittelyn työkaluna 20
5 Vaatimusmäärittely Digitaalinen viinikellari -sovellukselle 22
5.1 Vaatimusten kartoittaminen 23
5.1.1 Kilpailijavertailu 24
5.1.2 Asiakaskysely ja sovellusluonnos 24
5.2 Vaatimusten analysointi 26
5.2.1 Kilpailijavertailu 26
5.2.2 Asiakaskysely ja sovellusluonnos 27
5.3 Dokumentointi 29
5.4 Validointi ja priorisointi 30
6 Johtopäätökset 32
7 Yhteenveto 35
Lähteet 36

Kuvat ja taulukot

Kuva 1 Projektit ja liiketoiminta 2
Kuva 2 Vesiputousmalli 4
Kuva 3 Ketterän ohjelmistokehityksen julistus 7
Kuva 4 Vaatimusmäärittelyn vaiheet 13

Taulukko 1 Yhteenveto analysoiduista vastauksista 29
Taulukko 2 Dokumentoidut vaatimukset 30

Liitteet

Liite 1 Aineistonhallintasuunnitelma
Liite 2 Asiakaskysely: Digitaalinen viinikellari -sovelluksen toiminnallisuudet
Liite 3 Luonnos Digitaalinen viinikellari -sovelluksesta
Liite 4 Digitaalinen viinikellari -sovelluksen kehitysjono

1 Johdanto

Hyvä sovellus ei ole vain hyvää koodia ja hienoa teknistä toteutusta, vaan se vastaa asiakkaan tarpeeseen. Käyttäjä ei käytä sovelluksia siksi, että sovellukset on ohjelmoitu hyvin vaan siksi, että sovellusta käyttäessä käyttäjä pystyy tekemään jotain mitä käyttäjä haluaa tehdä. Asiakkaiden tarpeista juontavat ohjelmiston toiminnallisuudet ja ohjelmiston laatu ovat vaatimuksia ohjelmistolle ja ne ovat vaatimusmäärittelyn keskiössä. Vaatimusmäärittely prosessina huolehtii, että nämä tarpeet ja laatuvaatimukset kartoitetaan, analysoidaan, dokumentoidaan ja validoidaan. Onnistunut vaatimusmäärittely on avain onnistuneeseen projektiin ja menestyvään ohjelmistoon.

Nopeasti muuttuva ja kehittyvä maailma vaikeuttaa ohjelmiston vaatimusten määrittelyä. Pyytämättä ja yllättäen tulevat muutostarpeet ja uudet vaatimukset ovat tässä ajassa useille projekteille arkipäivää. Yksi ohjelmistokehityksen tunnistetuista haasteista onkin juuri vaatimusmäärittely. Miten pystytään tunnistamaan, keräämään ja määrittelemään halutut toiminnallisuudet ja laatuvaatimukset, jos maailma ympärillämme ja tätä kautta myös tarpeet ja vaatimukset muuttuvat nopeasti ja odottamatta?

Opinnäytetyön aihe nousi halusta ymmärtää mitä ohjelmistokehittämisessä vaatimusmäärittelyllä tarkoitetaan ja minkälainen rooli vaatimusmäärittelyllä on ketterässä ohjelmistokehityksessä. Opinnäytetyön käytännön työnä tehdään vaatimusmäärittely uudelle ohjelmistolle ketterän ohjelmistokehitykseen arvoja ja periaatteita noudattaen.

Opinnäytetyöstä rajataan ulos vaatimustenmäärittelyssä käytettyjen työkalujen läpikäynti ja esittely. Opinnäytetyön ulkopuolelle rajataan myös ohjelmistokehittämisen elinkaaren muut vaiheet kuin vaatimusmäärittely eli suunnittelu, toteutus, testaus ja käyttöönotto.

Opinnäytetyön tutkimuskysymykset:

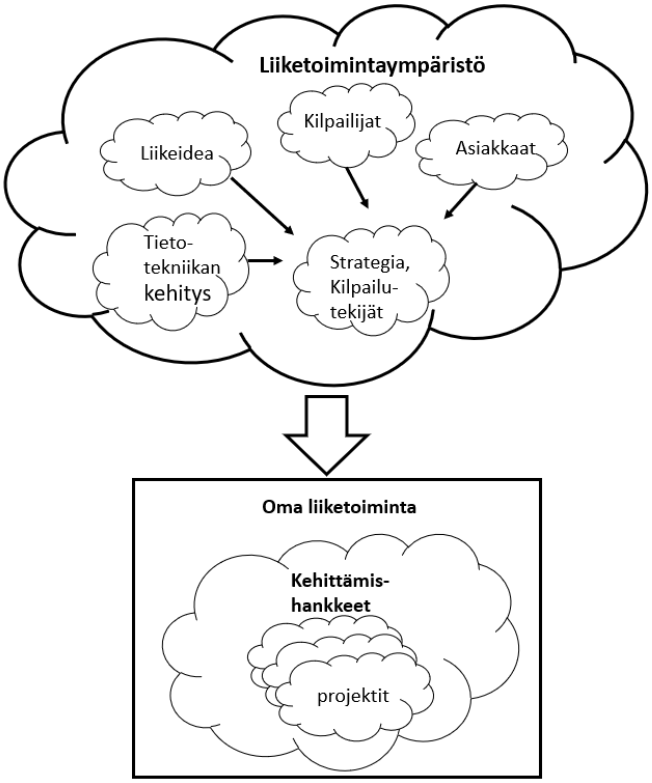
- Mitä tarkoitetaan vaatimusmäärittelyllä?
- Minkälainen rooli vaatimusmäärittelyllä on ketterässä ohjelmistokehityksessä?
- Soveltuuko ketterän ohjelmistokehityksen tapa tehdä vaatimusmäärittelyä pieneen projektiin?

2 Ohjelmistoprojektimallien synty ja ketteröityminen

Ohjelmistoprojektimallilla tarkoitetaan projektimallia, jolla hallinnoidaan ohjelmiston kehitystä ja projektin etenemistä. Projektimalli ei ota kantaa itse työn tekemiseen, vaan se on viitekehys, jonka avulla ohjataan työtä. Kun ensimmäisiä suuria ohjelmistoja alettiin kehittämään, huomattiin nopeasti tarve projektimallille, jonka avulla työtä pystyttäisiin ohjaamaan systemaattisesti. (Haikala & Mikkonen, 2011; Poimala & Tolvanen, n.d.)

Ohjelmistojen kehittäminen organisoidaan yleensä projektiksi (kuva 1). Ohjelmiston kehittämistä voidaan kutsua myös kehittämistoiminnaksi tai tuotekehitykseksi. Kehittäminen on konkreettista toimintaa, jolla tavoitellaan jonkin liiketoimintatavoitteen saavuttamista. Yritykset tarvitsevat jatkuvaa kehitystyötä monista eri syistä esimerkiksi: asiakkaiden tarpeiden ja mieltymysten ymmärtämiseksi, tuotteiden ja palveluiden parantamiseksi, tehokkuuden lisäämiseksi, ilmenneiden ongelmien ratkomiseksi, uusien tuotteiden ja palveluiden luomiseksi tai kasvun aikaansaamiseksi. (Haikala & Mikkonen, 2011; Ojasalo et al., 2020; Toikko & Rantanen, 2009)

Kuva 1 Projektit ja liiketoiminta (Haikala & Mikkonen, 2011)



Ohjelmistoprojekti koostuu yleensä projektimallista riippumatta vaatimusmäärittelystä, suunnittelusta, toteutuksesta, testauksesta ja käyttöönotosta. Vaatimusmäärittelyllä tarkoitetaan, nimensä mukaisesti, projektiin ja ohjelmistoon liittyvien vaatimusten eli esimerkiksi asiakastarpeiden ja laatuvaatimusten määrittelyä. Suunnittelulla tarkoitetaan kehitettävän järjestelmän teknistä suunnittelua. Suunnittelu perustuu yleensä vaatimusmäärittelyyn. Toteutusvaiheessa ohjelmisto toteutetaan eli koodataan ja testausvaiheessa toteutettu ohjelmisto testataan ja varmistetaan, että se toimii virheettömästi vaatimusten mukaisesti. Testauksen jälkeen ohjelmisto otetaan yleensä käyttöön. (Haikala & Mikkonen, 2011; Luukkainen, 2020)

Vuosien varrella on kehitetty lukuisia projektimalleja, joista osa on suunniteltu nimenomaan ohjelmistoprojektien hallintaan. Yksinkertaisimmissa projektimalleissa ohjelmointi on pääroolissa; ohjelmaa kasvatetaan ohjelmoimalla, muutetaan ja korjataan kunnes ohjelma vastaa haluttua. Monimutkaisissa ohjelmissa ja projekteissa ohjelman kehittämiseen tarvitaan kuitenkin yleensä paljon muutakin kuin vain ohjelmointia. Eri projektimallit eroavat toisistaan siinä, miten projektin vaiheita toteutetaan, missä vaiheessa ja minkälaisia rooleja projektissa on. (Haikala & Mikkonen, 2011)

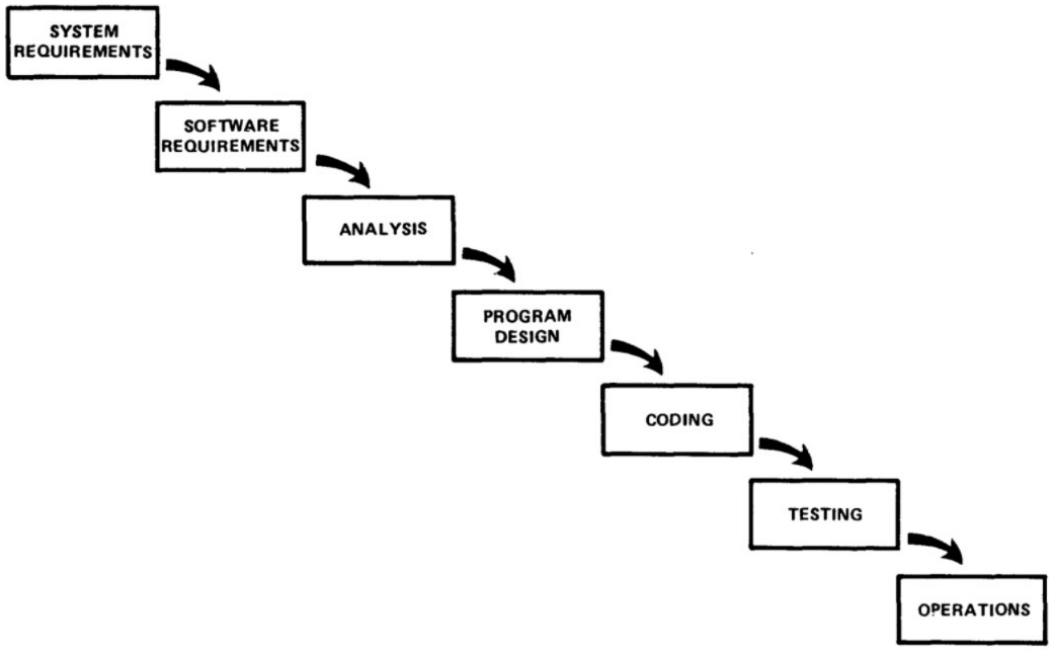
2.1 Vesiputousmallista iteratiivisten projektimallien yleistymiseen

Ajan myötä ohjelmistoja haluttiin alkaa kehittämään laajemmin ja samaan aikaan tietokoneiden teho kasvoi. Kehittämisen laajeneminen ja tietokoneiden tehon kasvu aiheutti uusia riskejä, joihin pyrittiin reagoimaan luomalla yhä enemmän strukturoituja ja kontrolloituja projektimalleja kehittämisen ja riskienhallinnan avuksi. Tämä johti lopulta Vesiputousmallin syntyyn 1970-luvulla. Vesiputousmallia pidetään ensimmäisenä ohjelmistokehityksen projektimallina ja se saavutti laajan suosion heti syntyessään. (Leffingwell, 2011; Taina, 2009)

Vesiputousmalli on lineaarinen eli suoraviivainen projektimalli, jossa projektin vaiheet seuraavat toisiaan (kuva 2). Vaatimusmäärittely ja suunnittelu tehdään projektin alussa, jonka jälkeen ohjelmisto toteutetaan näiden määritysten ja suunnitelmien pohjalta. Lopussa testauksella varmistetaan, että ohjelmisto toimii kuten se on määritelty ja suunniteltu.

Vesiputousmallissa projektin vaiheet perustuvat edellisen vaiheen työhön ja vaiheet toteutetaan yleensä eri tiimien toimesta. (Leffingwell, 2011)

Kuva 2 Vesiputousmalli (Royce, 1970)



Winston Roycea pidetään vesiputousmallin luojana, vaikka jo julkaisun yhteydessä Royce itse kritisoi esittelemäänsä projektimallia. Royce oli sitä mieltä, että vesiputousmalli ei sovi isoihin ohjelmistoprojekteihin. Royce esittelikin vesiputousmallin julkaisemisen yhteydessä myös vesiputousmallista parannetun version, jonka ensimmäinen vaihe on ohjelmiston prototyypin rakentamisen. Parannetussa vesiputousmallissa prototyypin ja siitä saadun palautteen jälkeen rakennettaisiin itse ohjelmisto. Roycen parannettu vesiputousmalli ei kuitenkaan saanut suosiota vaan alkuperäinen, Roycen itsensä kritisoima, vesiputousmallin jäi elämään. (Leffingwell, 2011)

Muutama vuosikymmen vesiputousmallin syntymisen ja sen käytön yleistymisen jälkeen projektien epäonnistumisia analysoivissa tilastoissa alkoi korostumaan vesiputousmallin haasteet. Vesiputousmallissa projektin alussa ohjelmisto suunnitellaan sekä määritellään ja nämä vaiheet muodostavat projektille pohjan. Näiden pohjalta usein projektille myös arviodaan kesto sekä kokonaiskulut ja ne muodostavat kehittämisee raamit. Nämä raamit sekä projektin alussa määritelty suunta aiheuttavat sen, että mahdollisiin projektin aikana ilmeneviin muuttuviin vaatimuksiin on vaikea ellei jopa mahdoton reakoida. Moni kokeekin,

että vesiputousmallin suurin haaste onkin kyky reagoida muutoksiin ja muuttuviin vaatimuksiin. (Leffingwell, 2011; Luukkainen, 2020)

Vesiputousmallissa on tärkeää, että kaikki toteutettavat vaatimukset on kerätty ja dokumentoitu projektin alussa, ennen suunnittelun aloittamista. Vesiputousmallissa myös vaatimusten dokumentointi korostuu, se tulee olla huolellisesti tehty, kattava ja ristiriidaton. Vesiputousmallissa vaatimusmäärittelyn raskas rooli on yksi syy, miksi kevyemmät menetelmät alkoivat saada suosiota 1990-luvulla. Jo tuolloin tunnistettiin, että asiakkaiden ja asiakkaiden edustajien on hyvin vaikea kertoa kaikkia vaatimuksia etukäteen. Asiakkaiden ja toimintaympäristön vaatimukset voivat myös muuttua hyvinkin nopealla aikataululla. (Luukkainen, 2020)

Vesiputousmallissa tunnistettujen haasteiden sekä ohjelmistokehittämisen välineiden ja tekniikoiden kehittymisen myötä alkoi syntymään tarve uusille projektimalleille. Tämä johti iteratiivisten prosessien yleistymiseen 1990 – luvuilla. Iteratiivisissa prosesseissa ohjelmistotuotanto jaettaan iteraatioihin joiden aikana ohjelmistoa kehitetään osissa. Vesiputousmallista poiketen ohjelmistoa ei määritellä ja suunnitella kokonaan alussa, vaan määrittely, suunnittely, toteutus ja testaus toteutetaan iteraatioiden aikana. Ohjelmisto rakentuu pala palalta ja asiakas pääsee näkemään iteraatioiden välissä valmistuneen version ohjelmasta. Tämä mahollistaa sen, että asiakkaan uusiin vaatimuksiin tai muutoksiin pystytään reagoimaan jo ennen kuin ohjelmisto on valmis. (Leffingwell, 2011; Luukkainen, 2020)

1990-luvulla yleistyneessä iteratiivisessa prosessien myötä pystyttiin reagoimaan paremmin muutoksiin, mutta yleisesti 1980 – 1990 – lukujen prosessimalleissa korostui yhä huolellinen suunnittelu, formaali laadunvalvonta ja tarkasti ohjattu prosessi. Vallalla olevat prosessimallit vastasivat isojen projektien tarpeita, kun taas pienille ja keskisuurille projekteilla mallit olivat yhä liian raskaita ja jäykkiä. Iteratiivisissä prosesseissa myös yksilön merkitys projektille nähtiin perinteisten projektimallien tapaan merkityksettömänä. (Leffingwell, 2011; Luukkainen, 2020)

1990-luvusta alkaen ohjelmistokehityksessä alkoin näkymään yhä kevyempiä ja ketterämpiä malleja, työkaluja ja prosesseja. Vasiputousmallille tyypillinen etukäteen tehtävä määrittely,

suunnittelu ja raskas dokumentointi alkoi saamaan kilpailijoita. Nämä kevyemmät ja ketterämmät prosessit uskoivat, että oikeilla kehittämisen työkaluilla ja menetelmillä oli kustannustehokkaampaa tuottaa ohjelmistoa niin, että asiakkaille vietiin ohjelmisto käyttöön hyvin aikaisessa vaiheessa ja palautteen perusteella ohjelmistoa korjattaisiin. Nämä kevyemmät ja ketterämmät käytännöt katsotaan syntyneen ohjelmistoalan tahdosta vastata alan haasteisiin eli halusta pystyä reagoimaan muutoksiin paremmin, sekä kehittämään laadukkaampia ohjelmistoja kustannustehokkaammin ja kevyemmin. (Leffingwell, 2011)

Ohjelmistokehittäminen on aineettomien ideoiden kehittämistä, jotka kuvataan binaarisenä koodina. Koska kyseessä ei ole fyysisten tuotteiden kehittäminen, ohjelmiston vaatimusten määrittely ja hallinta nousee keskiöön. Ajan myötä ohjelmistot joita kehitettiin kasvoivat isommiksi ja menetelmät, joilla yritettiin kontoroloida ja hallita kehitysprojekteja, muuttuivat raskaimmiksi ja raskaimmiksi. Kehitys alkoi tahattomasti hidastamaan sitä asiaa, jota haluttiin nopeuttaa – kykyä toimittaa laadukkaita ja arvoa tuottavia ohjelmistoja. Tarve yhä kevyemmille ja ketterämmille projektimalleille jatkoi kasvamistaa. (Leffingwell, 2011)

2.2 Ketterän ohjelmistokehityksen synty

Meitä ympäröivä maailma muuttuu yhä nopeammin. Yritysen kyky reagoida muutoksiin ja muutosten nopea omaksuminen vaikuttavat yrityksen mahdollisuuksiin menestyä. Nopeasti muuttuvassa maailmassa perinteisten ohjelmistoprojektimallien koetaan epäonnistuneen toistuvasti tuottamaan ohjelmistoa sovitussa ajassa. Ketterä ohjelmistokehitys syntyi vastaamaan muun muassa tähän haasteeseen. (Agile Alliance, 2021; Ojasalo et al., 2020)

Ketterä ohjelmistokehitys syntyi tahtotilasta löytää parempia tapoja vastata tunnistettuihin kehittämisen haasteisiin ja kehittää parempaa ohjelmistoa. Vuonna 2001 ryhmä ketterien menetelmien sekä työkalujen ja prosessien edelläkävijöitä kokoontui yhteen ja tapaamisen pohjalta syntyi Agile manifesto – ketterän ohjelmistokehityksen julistus. Ketterän ohjelmistokehityksen julistus on yhteenveto näkemyksestä miten ohjelmistoja voidaan tuottaa ja kehittää paremmin. Julistus tiivistää ketterän ohjelmistokehityksen arvot ja luo perustan ketterille projektimalleille ja työkaluille. (Agile Alliance, 2021; Leffingwell, 2011; Luukkainen, 2020)

2.3 Ketterä ohjelmistokehitys

Ketterän ohjelmistokehityksen julistus koostuu neljästä pääkohdasta, joissa kuvataan ketterän arvojen kautta tärkeimmät asiat millä mahdollistetaan onnistunut ohjelmistokehittäminen. Julistuksessa yksilöitä ja kanssakäymistä arvostetaan yli prosessien ja työkalujen, toimivaa ohjelmistoa arvostetaan yli kattavan dokumentaation, asiakasyhteistyötä arvostetaan yli sopimusneuvotteluiden ja kykyä reagoida muutokseen arvostetaan yli suunnitelmassa pysymisen (kuva 3). (Agile Alliance, 2001c; Luukkainen, 2020)

Kuva 3 Ketterän ohjelmistokehityksen julistus (Agile Alliance, 2001a)

Muutama kuukausia ketterän ohjelmistokehityksen julistuksen jälkeen julkaistiin 12 periaatetta. Ketterän ohjelmistokehittämisen 12 periaatetta ovat:

1. ”Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.
2. Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.
3. Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.
4. Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.

5. Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.
6. Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.
7. Toimiva ohjelmisto on edistymisen ensisijainen mittari.
8. Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.
9. Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.
10. Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.
11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.
12. Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti. ”
    (Agile Alliance, 2001b, 2021)

12 periaatteen mukaisesti, ketterän ohjelmistokehityksen ensisijainen mittari on tuottaa asiakkaalle laadukkaita, toimivia ja asiakkaan tarpeeseen soveltuvia ohjelmistoja. Ketterässä, toisin kuin perinteisissä projektimalleissa, muuttuvat vaatimukset koetaan osana prosessia ja niihin reagoidaan. Muuttuviin vaatimuksiin reagomalla varmistetaan, että kehitettävä ohjelmisto vastaa asiakkaan tarpeita. Yhteistyö ja itseorgaisoituvat tiimit, jotka koostuvat liiketoiminnanedustajista ja ohjelmistokehittäjistä, mahdollistavat ketterässä laadukkaan ohjelmiston tuottamisen. Yksilön rooli onnistuneen ohjelmistokehittämisen kannalta on merkittävä. (Saddington, 2012)

Ketterän ohjelmistokehityksen julistus ja 12 periaatetta luovat ketterälle ohjelmistokehittämiselle perustan ja raamit. Ketterä ohjelmistokehitys, julistus ja periaatteet eivät kuitenkaan ole yksinään projektimalleja, vaan ne muodostavat käsitteen, sateenvarjon, jonka alle sijoittuu projektimalleja ja käytäntöjä. Agile Alliance kuvailee, että ketterä ohjelmistokehitys on enemmänkin mielentila tai adjektiivi, joka auttaa toimimaan epävarmuuden kanssa. (Agile Alliance, 2021)

Ketterässä ohjelmistokehityksessä tavoite on tuottaa oikeita asioita ja tuotteita oikeaan aikaan. Asiakas osallistetaan tuotteen kehittämiseen ja näin varmistetaan todellinen ymmärrys asiakkaan tarpeista. (Agile Alliance, 2021; Auer et al., 2013)

2.4 Ketterän ohjelmistokehityksen projektimallit

Ketterät ohjelmistokehityksen projektimallit perustuvat ketterän ohjelmistokehityksen arvoihin ja periaatteisiin. Ketterien projektimallien lisäksi on olemassa ketteriä käytäntöjä, joita voidaan toteuttaa projektimallien sisällä tai erikseen. Ketteriä käytäntöjä ovat muun muassa: pariohjelmointi, testivetoinen kehittäminen, päivittäiset tilannepalaverit, iteraation suunnittelupalaveri ja iteraatio. Ketterän ohjelmistokehityksen projektimalleja ovat muun muassa: Scrum, Extreme Programming (XP) ja Feature-Driven Development (FDD). Vuonna 2020 ketterän ohjelmistokehityksen projektimallien käyttäjistä 58 % hyödynsi Scrum-projektimallia. (Agile Alliance, 2021; Digital.ai, n.d.)

Ketterän ohjelmistokehityksen projektimalleille on tyypillistä iteratiivinen eli syklimäinen kehittäminen. Kehittämistä tehdään sovitun pituisten iteraatioiden aikana, jonka aikana tuotteesta pyritään saaman versio valmiiksi. Iteraation päätteeksi valmistuneesta ohjelmiston versiosta pyydetään asiakkailta palautetta ja palautteen perusteella kehittämistä jatketaan seuraavassa iteraatiossa. Iteraatiota voidaan kutsua myös Sprintiksi. (Poimala & Tolvanen, n.d.; Scrum.org, 2021)

Iteraation käynnistää yleensä iteraation suunnittelupalaveri, jossa valitaan iteraatioon mukaan sovittu määrä käyttäjätarinoita, joihin päivitetään yksityiskohtaiset vaatimukset ja hyväksymiskriteerit. Mukaan valitut käyttäjätarinat toteutetaan iteraation aikana. Tehtävää työtä priorisoidaan sen mukaan, millä on suurin arvo asiakkaalle. Iteraation tavoite on aina tuottaa toiminnallisuuksia, joiden avulla pystytään tuottamaan asiakkaalle arvoa. (Leffingwell, 2011; Luukkainen, 2020)

Iteraation aikana tiimi seuraa iteraatioon mukaan otettujen tehtävien edistymistä päivittäisessä tilannepalaverissa, joita voidaan kutsua myös Dailyksi. Palaverin kesto on aina 15 minuuttia, riippumatta osallistujien määrästä. Palaverin tarkoitus on käydä läpi jokaisen

tiimiläisen osalta tehtävien edistyminen sekä mahdolliset esteet edistymiselle. (Leffingwell, 2011)

Vaikka ketterä ohjelmistokehitys ottaa kantaa ainoastaan ohjelmistokehitykseen ja yrittää ratkoa siinä havaittuja haasteita, ketterää ideologiaa voi hyödyntää myös muissa toiminnoissa. Tärkeää on pitää mielessä ketterän ohjelmistokehityksen julistus ja 12 periaatetta ja miettiä kuinka tulisi toimia, jotta pystyttäisiin saamaan aikaan tuottamaan uusia asioita, vastaamaan muutokseen ja toimimaan epävarmuudessa. (Agile Alliance, 2021)

3 Vaatimusmäärittely

Kuten aikaisemmin todettiin, ohjelmistoprojekti koostuu yleensä vaatimusmäärittelystä, suunnittelusta, toteutuksesta, testauksesta ja käyttöönotosta. Vaatimusmäärittelyllä tarkoitetaan kehitettävään ohjelmistoon liittyvien toiminnallisuuksien, ominaisuuksien ja vaatimusten määrittelyä. Jotta yritys pystyy kehittämään menestyviä ohjelmistoja, tulee yrityksen ymmärtää minkälaisia vaatimuksia käyttäjillä, sidosryhmillä ja muilla sovelluksilla on ohjelmistolle sekä miten ohjelmiston tulisi toimia ja mihin tarpeeseen vastata. (Haikala & Mikkonen, 2011; Leffingwell, 2011; Luukkainen, 2020; Ojasalo et al., 2020)

Projektien epäonnistumisista arviolta 60–80 % johtuu vaatimusmäärittelyn epäonnistumisesta. Kokeneet kehittäjät tietävät, että vaatimusten määrittely ja hallinta on yksi suurimmista haasteista ohjelmistojen toteutuksessa ja kehittämisessä. Kunnolla hoidettu vaatimusmäärittely on onnistuneen ohjelmistokehittämisen perusedellytys. Vaatimusmäärittelyllä varmistetaan, että kehitettävä ohjelmisto tuottaa käyttäjilleen arvoa. Ilman arvon tuottamista, ohjelmisto on vain koodistoa (Haikala & Mikkonen, 2011; Leffingwell, 2011)

Hyvän vaatimuksen ominaisuuksia ovat virheettömyys ja selkeys. Näiden lisäksi vaatimus tulee olla testattava ja jäljitettävä. Testattavalla tarkoitetaan, että vaatimus on pystyttävä testaamaan, eli varmistamaan onko vaatimus toteutettu vaaditulla tavalla. Vaatimukset luokitellaan yleensä kahteen eri luokkaa: toiminnalliset ja ei-toiminnalliset. Toiminnalliset vaatimukset kuvaavat ohjelmiston toimintoja, kun taas ei-toiminnalliset vaatimukset liittyvät järjestelmän sisäisiin ja teknisiin ominaisuuksiin ja vaatimuksiin. (Haikala & Mikkonen, 2011; Leppänen, 2021)

3.1 Toiminnalliset ja ei-toiminnalliset vaatimukset

Toiminnalliset vaatimukset ovat nimensä mukaisesti ohjelmiston toimintoja eli asioita, joita ohjelmistolla tulee pystyä tekemään. Toiminnallisia vaatimuksia ovat esimerkiksi: asiakas voi rekisteröityä käyttäjäksi ja kirjautunut asiakas näkee ostoshistoriansa. Toiminnalliset vaatimukset kuvaavat yksityiskohtaisesti käyttäjän vuorovaikutuksen ohjelmiston kanssa eli

miten ja mitä syötteitä käyttäjä antaa ohjelmistolle ja miten ohjelmisto reagoi näihin syötteisiin. (Luukkainen, 2020)

Toiminnallisten vaatimusten lisäksi on olemassa ei-toiminnallisia vaatimuksia, jotka määrittelevät rajotteita ja reunaehtoja ohjelmiston tekniselle toteutukselle. Ei-toiminnalliset vaatimukset voidaan jakaa kahteen ryhmään: laatuvaatimukset ja toimintaympäristön rajoitteet. Ei-toiminnallisia laatuvaatimuksia ovat esimerkiksi: tietoturva, käytettävyys, skaalattavuus, suorituskyky, stabiilisuus ja ylläpidettävyys. Ei-toiminnallisia toimintaympäristön rajoitteita ovat esimerkiksi: lait ja standardit sekä käyttäjäympäristö. Ei-toiminnalliset vaatimukset eivät välttämättä ole käyttäjien. havaittavissa. Ei-toiminnalliset vaatimukset vaikuttavat kriittisesti asiakasarvon ja asiakaskokemuksen tuottamiseen ja ne tuleekin ottaa huomioon koko ohjelmiston elinkaaren ajan, ei vain kehittämisvaiheessa. (Leppänen, 2021; Luukkainen, 2020)

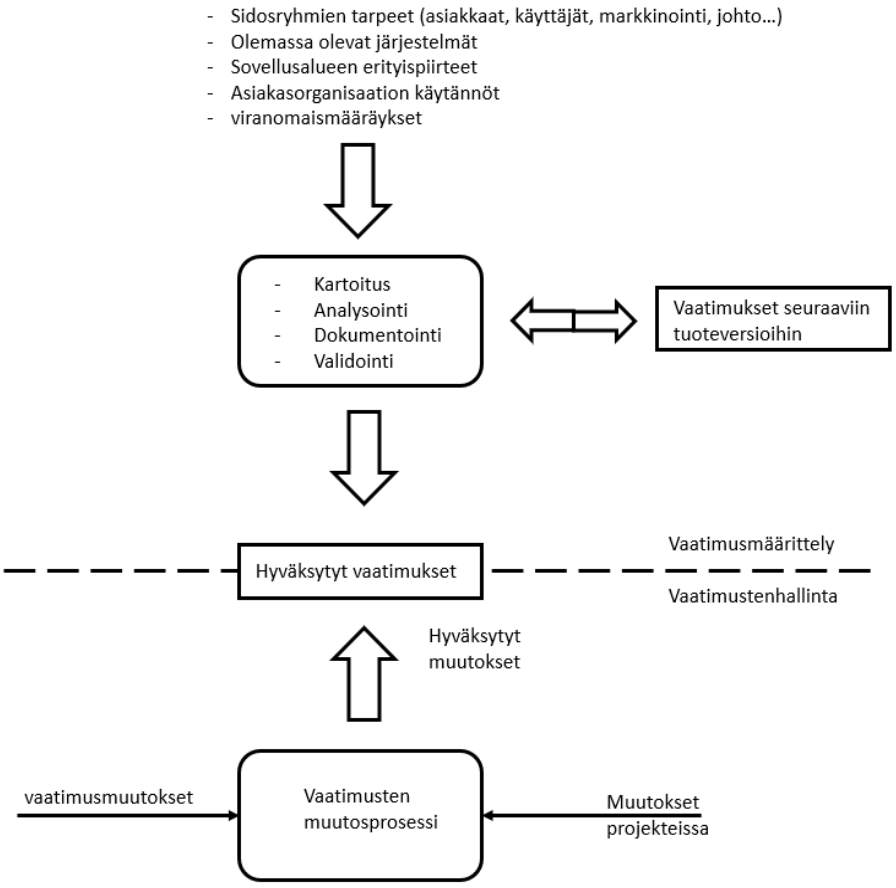
Toiminnalliset vaatimukset ovat yleensä ohjelmiston yksittäisiä toimintoja, kun taas ei-toiminnalliset vaatimukset koskevat koko ohjelmistoa, ohjelmiston perustasta alkaen. Ei-toiminnallisten vaatimusten vaikutusta koko ohjelmistoon voidaan kuvata seuraavasti: jos halutaan, että ohjelmistossa pystyy asioimaan yhtä aikaisesti miljoona käyttäjää, tulee tämä ottaa huomioon jo ohjelmiston perustaa suunnitellessa. Jos ohjelmisto rakennetaan ensin kestämään muutamien kymmenien käyttäjien asiointia, tarvittavien muutosten tekeminen jälkikäteen voi vaatia huomattavaa työtä. (Luukkainen, 2020)

Toiminnallisten vaatimusten määrittely koetaan usein helpommaksi kuin ei-toiminnallisten vaatimusten. Ei-toiminnallisten vaatimusten määrittely vaatii usein teknistä ja liiketoiminnallista osaamista. Ei-toiminnallisten ja toiminnallisten vaatimusten erot voi yksinkertaistaa seuraavasti: toiminnalliset vaatimukset määrittelevät ohjelmiston liiketoiminnallisen hyötypotentiaalin, kun taas ei-toiminnalliset vaatimukset määrittelevät ohjelmiston luotettavuuden, kustannustehokkuuden sekä teknisen mukautuvuuden tämän hetken ja tulevaisuuden tarpeisiin. (Leppänen, 2021)

3.2 Vaatimusmäärittelyn vaiheet

Vaatimusten määrittely etenee yleensä spiraalimaisesti, tarkentuen jokaisella kierroksella. Ensin vaatimukset kartoitetaa ja analysoidaan, jonka jälkeen ne dokumentoidaan ja validoidaan (kuva 4). (Luukkainen, 2020)

Kuva 4 Vaatimusmäärittelyn vaiheet (Haikala & Mikkonen, 2011)



Kehitettävä ohjelmisto, projektimalli sekä organisaatio, jossa kehittämistä tehdään, vaikuttavat siihen miten vaatimusmäärittely toteutetaan. Riippumatta siitä miten vaatimusmäärittely toteutetaan, tärkeää on aina huolehtia, että vaatimusmäärittelyssä on mukana asiakas ja asiakkaa edustaja. Mikäli vaatimukset ja kehittäminen eivät perustu asiakkaiden, käyttäjien ja sidosryhmien vaatimuksiin ja tarpeisiin, kehitysprojekti tulee hyvin todennäköisesti epäonnistumaan. (Juvonen, 2018; Luukkainen, 2020)

3.2.1 Vaatimusten kartoittaminen ja analysointi

Aivan ensimmäisenä vaatimusmäärittely prosessissa selvitetään sidosryhmät, eli tahot, jotka ovat suoraan tai epäsuoraan tekemisissä kehitettävän ohjelmiston kanssa. Sidosryhmiä voi olla esimerkiksi ohjelmiston loppukäyttäjät, asiakkaiden edustajat yrityksessä sekä mahdolliset tahot, jotka vastaavat ohjelmistosta, johon kehitettävä ohjelmisto tullaan integroimaan. Työpaikan hiljaista ja kokemuksellista ammattitietoa tulee aina tarkastella kriittisesti ja varmistaa, että tämä vastaa kehittämisen tahtotilaa. (Luukkainen, 2020)

Kun sidosryhmät on löydetty, alkaa vaatimusten kartoittaminen. Vaatimusten kartoittamisessa eli löytämisessä voidaan käyttää erilaisia menetelmiä, esimerkiksi haastatteluja, vanhojen ohjelmistojen läpikäyntiä tai työpajoja, johon osallistuvat kehitystiimi, ohjelmiston loppukäyttäjät ja asiakkaiden edustajat. (Luukkainen, 2010, 2020)

Vaatimuksia voidaan kartoittaa esimerkiksi seuraavasti:

- Ohjelmiston käyttäjistä luodaan käyttäjäroolit ja niiden perusteella käydään läpi mitä tyypillisiä käyttöskenaarioita näillä käyttäjärooleilla mahdollisesti olisi.
- Tehdään ohjelmiston käyttöliittymästä luonnos tai esimerkiksi paperinen prototyyppi, jonka avulla asiakas voi tarkentaa näkemystään toiminnallisuuksista ja löytää uusi vaatimuksia.
- Seurataan vanhan ohjelmiston käyttäjien työskentelyä. Vanhaan prosessiin tulee kuitenkin suhtautua kriittisesti ja tarkastella voisiko prosessia parantaa uuden ohjelmiston avulla.
  (Luukkainen, 2020)

Vaatimusten kartoittamisen jälkeen vaatimukset analysoidaan. Vaatimusten analysoinnissa varmistetaan, että vaatimukset ovat riittävän kattavia ja ne on testattavissa. Tämän lisäksi varmistetaan, että vaatimuksilla ei ole ristiriitoja keskenään ja ne on mahdollista ja järkevää toteuttaa. Vaatimuksien tulee olla myös konkreettisia ja ne tulee olla testattavissa. Testattavalla tarkoitetaan, että vaatimus pitää pystyä toteutuksen jälkeen testaamaan ja varmistamaan, että se on toteutettu vaatimuksen mukaisesti. (Jokela, 2009; Luukkainen, 2020; Ojasalo et al., 2020)

3.2.2 Vaatimusten dokumentointi ja validointi

Kartoitetut ja analysoidut vaatimukset kuvataan usein erilaisina käyttötapauksina tai tekstinä. Vaatimusten dokumentointi on tärkeä osa vaatimusmäärittelyn prosessia. Vaatimus tulee olla dokumentoitu, jotta henkilö, joka konkreettisesti tekee ohjelmiston eli koodaa sen, pystyy rakentamaan ohjelmiston toimimaan vaatimusten mukaisesti. Perinteisissä projektimalleissa vaatimusmäärittelyn tuloksena syntyy yleensä dokumentti, johon on kerätty kaikki toiminnalliset ja ei-toiminnalliset vaatimukset. (Haikala & Mikkonen, 2011; Luukkainen, 2020)

Dokumentoituja vaatimuksia tarvitaan myös ohjelmiston testausvaiheessa, jolloin muun muassa testataan, että ohjelmisto toimii kuten se on määritelty toimivan. Vaatimusmäärittely dokumentti voi toimia myös joissakin projektimalleissa, kuten vesiputousmallissa asiakkaan edustajan ja ohjelmiston kehittäjän välisenä sopimuksena. (Luukkainen, 2020)

Viimeiseksi vaatimukset validoidaan eli varmistetaan, että ne ovat sellaisia toiminnallisuuksia ja vaatimuksia, jotka halutaan toteuttaa. Kartoitetut, analysoidut, dokumentoidut ja validoidut vaatimukset kuvaavat minkälainen kehitettävästä ohjelmistosta tulee. Vaatimuksia pitää pystyä kuitenkin myös jälkikäteen muokkaamaan ja lisäämään. Hyväksyttyjen vaatimusten muokkaamista tai lisäämistä kutsutaan vaatimustenhallinnan prosessiksi. (Luukkainen, 2020)

4 Vaatimusmäärittely ketterässä ohjelmistokehityksessä

Jo hyvin aikaisessa vaiheessa ohjelmistokehityksen historiaa huomattiin, että asiakkaiden on hyvin haastavaa, ellei mahdotonta, ilmaista kaikkia tarpeitaan ja vaatimuksiaan etukäteen. Myös toimintaympäristö, markkinat ja asiakkaiden mieltymykset voivat muuttua hyvinkin nopeasti, joka tarkoittaa sitä, että jos projekti ei pysty reagoimaan muuttuviin vaatimuksiin, ohjelmisto voi olla jo vanha valmistuessaan. (Leffingwell, 2011; Luukkainen, 2020)

Jos täydellistä listaa vaatimuksista ei ole mahdollista selvittää etukäteen, miten vaatimusmäärittelyä tulisi sitten tehdä? Ketterän ohjelmistokehityksen vastaus tähän haasteeseen on muuttaa tapaa, jolla vaatimusmäärittelyä tehdään. Sen sijaan, että tehdään raskaita investointeja, että saadaan kerättyä kaikki yksityiskohtaiset vaatimukset etukäteen, ketterä kannustaa yrityksiä tekemään investointeja prosesseihin ja työkaluihin, joiden avulla asiakkaiden käytössä olevien ohjelmistojen huonot toiminnot, ratkaisut ja virheet pystytään löytämään ja korjaamaan nopealla aikataululla. (Leffingwell, 2011)

Ketterässä ohjelmistokehityksessä vaatimusmäärittelyyn suhtaudutaan eri lailla kuin perinteisissä projektimalleissa, esimerkiksi vesiputousmallissa. Sen sijaan, että projektin alkuvaiheessa keskityttäisiin yksityiskohtaiseen vaatimusten määrittelyyn, ketterässä vältetään etukäteen tehtävää yksityiskohtaista määrittelyä ja suunnittelua aina kun mahdollista. Ketterän ohjelmistokehityksen erilainen suhtautuminen vaatimusmäärittelyyn tulee esiin jo ketterän julistuksessa: ”Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi”. Ketterässä ohjelmistokehityksessä vaatimusmäärittely muuttuu joustavaksi. (Leffingwell, 2007, 2011)

Ketterässä ohjelmistokehityksessä perusolettamuksena on, että muutoksia tulee aina, riippumatta halutaanko tai odotetaanko niitä. Ketterässä ohjelmistokehityksessä ajatuksena onkin, että vaatimuksia ei kannata projektin alussa lyödä lukkoon tai käyttää huomattavaa määrää aikaa niiden määrittelyyn, vaan tehdä vaatimusmäärittely sillä tasolla, että sen perusteella kehitystyö voidaan aloittaa. Ketterässä ohjelmistokehityksessä vaatimusmäärittely tapahtuu ja tarkentuu paloissa kehitysjonon muodostamisen ja iteraatioiden aikana. (Juvonen, 2018; Leffingwell, 2011; Luukkainen, 2020)

Perinteisissä projektimalleissa kuten vesiputousmallissa vaatimusmäärittely toteutetaan projektin alussa, ennen suunnittelun aloittamista. Ajatuksena on, että kaikki vaatimukset kerätään ja huolellisesti dokumentoidaan heti projektin alussa ja tämän jälkeen nämä kerätyt ja dokumentoidut vaatimukset toteutetaan. Koko projekti perustuu näille jo etukäteen määritellyille vaatimuksille. Perinteisissä projektimalleissa tarve muuttaa vaatimuksia onkin harvoin mahdollista tai muutokset aiheuttavat projektin venymisen tai pahimmassa tapauksessa projektin epäonnistumisen. (Luukkainen, 2020)

Kun ketterän ohjelmistokehityksen tapaa tehdä vaatimusmäärittelyä alettiin hyödyntämään isoissa ohjelmistoprojekteissa, huomattiin ongelmia. Vaikka järjestelmällistä kehitysjonoa on helppo lukea, priorisoida ja hallita, se ei välttämättä taivu monimutkaiseen analyysiin, joita laajat ohjelmistot tarvitsevat. Vaikka käyttäjätarinat varmistavat asiakaslähtöisen kehittämisen, niissä on ei-toiminnallisten vaatimusten määrittelyssä ja dokumentoinnissa haasteita. (Behutiye et al., 2017; Leffingwell, 2011)

Kun projekti koko kasvaa, kommunikoinnista tulee haastavaa ja väärinymmärrysten mahdollisuus kasvaa. Näiden riskien hallinta vaatii myös uusi tapoja ja työkaluja vaatimusmäärittelyyn. Vaikka ketterä ohjelmistokehitys ohjaa hyödyntämään kevyitä tekniikoita, se samalla muistuttaa, että vaatimusmäärittelyssä ei kannata jättää huomiotta tekniikoita, jotka ovat aikaisemmin auttaneet hallinnoimaan monimutkaisuuksia. Tiimin vastuulla on valita käyttöönsä parhaat työkalut, joiden avulla pystytään tuottamaan ohjelmistoja ketterän arvojen ja periaatteiden mukaisesti. (Leffingwell, 2011)

Ketterä ohjelmistokehitys ohjaa välttämään etukäteen tehtävää suunnittelua ja määrittelyä aina kun se on mahdollista. Kuitenkin kehitettävien ohjelmien tulee aina toimia ja olla luotettavia. Jos käyttäjätarina ja luonnollinen kieli ei riitä vaatimustenmäärittelyyn, tulee kehittävän tiimin hyödyntää vaatimusmäärittelyssä tekniikoita ja työvälineitä, jotka pienentävät väärinymmärryksen ja epäonnistumisen riskiä. (Leffingwell, 2011)

4.1 Käyttäjätarina

Ketterässä ohjelmistokehityksessä toiminnallisten vaatimusten määrittelyyn käytetään usein käyttäjätarinoita. Käyttäjätarinoiden hyödyntäminen tarjoaa kevyen menetelmän määritellä

ja hallita vaatimuksia, samalla kun varmistetaan, että työ keskittyy asiakasarvon tuottamiseen, ei pelkkiin toiminnallisiin rakenteisiin. (Leffingwell, 2011)

Käyttäjätarinan tehtävä on kuljettaa asiakkaan vaatimukset koko kehittämisprosessin läpi määrittelystä, ohjelmointiin ja tuotantoon. Käyttäjätarinat ovat työkalu, joilla määritellään ohjelman toiminnot niin, että kaikki kehittämiseen osallistuvat ymmärtävät ne ja väärinymmärryksiä ei synny. Poiketen perinteisestä vaatimusmäärittelystä, käyttäjätarina on lyhyt lause, joka kuvaa yleensä käyttäjän näkökulmasta, miten käyttäjä haluaa ohjelmiston toimivan. Käyttäjätarinan runko on aina sama: ”As a <user>, I can <activity> so that <business value>”. (Leffingwell, 2011)

Käyttäjätarina on ketterässä ohjelmistokehityksessä työn yksikkö. Jokaisella käyttäjätarinalla on lyhyt intensiivinen kehittämisen elinkaari. Tiimi määrittelee, rakentaa ja testaa suunnitellun määrän käyttäjätarinoita iteraation aikana pystyäkseen tuottamaan asiakkailleen arvoa. (Leffingwell, 2011)

Ei-toiminnallisia vaatimuksia ei ole välttämättä järkevää esittää käyttäjätarinamuodossa. Tämä johtuu siitä, että kaikki ei-toiminnalliset vaatimukset eivät toimi samalla lailla keskenään tai ne eivät toimi samalla lailla kuin toiminnalliset vaatimukset. Osa ei-toiminnallisista vaatimuksista tulee testata vain yhden kerran, kun taas osa ei-toiminnallisista vaatimuksista pitää testata kehittämisen edetessä uudelleen ja uudelleen varmistuakseen, että uudet toiminnot eivät ole vaikuttaneet negatiivisesti ohjelmiston laatuun tai toiminnallisuuksiin. Sitä kuvataanko ei-toiminnalliset vaatimukset käyttäjätarinoiden avulla vai ei, ei ole merkityksellistä. Merkityksellistä on, että tiimi huomioi ei-toiminnalliset vaatimukset ja rakentaa tiimille sopivan tavan esittää ja huolehtia niistä. (Leffingwell, 2011)

4.2 Dokumentaatio

Dokumentaatio mielletään tärkeäksi osaksi vaatimusmäärittelyä. Ketterässä ohjelmistokehityksessä toimiva ohjelmisto nostetaan kuitenkin arvona yli dokumentaation. Tämä ei kuitenkaan tarkoita sitä, että dokumentaatiolla ole olisi ketterässä ohjelmistokehityksessä minkäänlaista roolia. Ketterässä dokumentaation määrää ja tapaa,

miten dokumentointia tehdään, ohjataan tarkastelemaan kriittisesti. Myös dokumentaatiota voidaan tehdä ketterästi. Tulee vain miettiä mitä dokumentteja aidosti tarvitaan ja miten dokumentointi toteutetaan, jotta se palvelee parhaiten tavoitetta toimittaa arvoa tuottavia versioita ohjelmistosta. Ketterä muistuttaa että, yksinään dokumentaatio ei ole itseisarvo, samalla lailla kuin yksinään vaatimusmäärittelykään ei ole itseisarvo. Tärkeää on huolehtia, että kaikki kehittämisprosessissa käytetyt välineet tukevat ketterän arvoja ja periaatteita. (Auer et al., 2013; Leffingwell, 2011)

Ketterässä ymmärretään, että dokumentaatio on työväline, joka auttaa muun muassa keräämään, jäsentämään ja dokumentoimaan asioita. Ketterässä kuitenkin etukäteen tehtävä yksityiskohtainen vaatimusmäärittelydokumentti on menettänyt arvonsa. Tämä ei kuitenkaan tarkoita, että ketterässä unohdettaisiin vaatimusten dokumentointi kokonaan. Ketterässä kannustetaan hyödyntämään kevyempiä työkaluja vaatimusten dokumentoinnissa esimerkiksi käyttäjätarinoita. Näiden kevyempien työkalujen, kuten käyttäjätarinoiden avulla pyritään välttämään liiallista investointia etukäteen tehtävään vaatimusmäärittelyyn, koska hyvin todennäköisesti vaatimukset tulevat muuttumaan. Ketterässäkin tunnistetaan, että dokumentointi on tärkeää, varsinkin niissä tilanteissa, kun kasvotusten tapahtuva kommunikointi ei ole mahdollista tai jos päätöksiä ja löydöksiä tullaan hyödyntämään pidemmän ajan sisällä. (Leffingwell, 2011)

4.3 Kehitysjono

Tiimin kehitysjono koostuu kaikista tunnistetuista käyttäjätarinoista, jotka pitää toteuttaa ennen kuin ohjelmisto on valmis. Toisin kuin perinteisessä vaatimusmäärittelyssä, kehitysjono ei kuitenkaan sisällä yksityiskohtaista listaa tuotteen vaatimuksista. Ketterässä myös tunnistetaan, että kehitysjono elää ja täydentyy iteraatioiden myötä. (Leffingwell, 2011)

Kun käyttäjätarina nostetaan kehitysjonosta iteraatioon toteutettavaksi, suunnitellaan käyttäjätarinan toteutus ja määritellään sille hyväksymiskriteerit. Toisin kuin esimerkiksi vesiputousmallissa tarkkoja yksityiskohtaisia vaatimuksia ei siis määritellä etukäteen, vaan yksityiskohdat määritellään siinä vaiheessa, kun käyttäjätarina otetaan toteutukseen. Näin

varmistetaan, että yksityiskohtaiset tiedot vaatimuksista ovat ajan tasalla eivätkä ne ole vanhentuneet, kun toteutus aloitetaan. (Auer et al., 2013; Luukkainen, 2020)

Kehitysjonon sisällön eli käyttäjätarinoiden työstämiseen osallistuu aina kehittävä tiimi ja tuotteen omistaja. Tämän lisäksi työstämiseen voi osallistua eri asiantuntijoita esimerkiksi käytettävyys-, tietoturva-, tai asiakaskokemusasiantuntijoita. Työstämisessä tärkeintä on ensimmäiseksi ymmärtää mitä halutaan tehdä ja mitä lisäarvoa tekeminen tuo asiakkaalle. Työstäminen voi myös päättyä siihen, että tekeminen ei tuo asiakkaalle lisäarvoa, joka tarkoittaa sitä, että ominaisuutta ei lähdetä toteuttamaan. (Auer et al., 2013)

Kehitysjonon hallinta on osa tiimin vaatimustenhallintaan. Ketterässä ohjelmistokehityksessä tunnistetaan ja hyväksytään, että uusia vaatimuksia ja vaatimusten muutoksia tulee, pyytämättä ja yllätyksenä ja niihin reagointi on avainasemassa onnistuneen ohjelmistokehittämisen kannalta. Tiimissä työskentelevän tuotteen omistajan tehtävänä on hallinnoida ja priorisoida tiimin kehitysjonossa olevia tehtäviä niin, että tärkeimmät eli asiakkaille eniten arvoa tuottavat toiminnallisuudet toteutetaan ensimmäisenä. Näin varmistetaan, että ohjelmisto tuottaa alusta lähtien mahdollisimman paljon arvoa käyttäjilleen. (Auer et al., 2013)

4.4 Iteraatio vaatimusmäärittelyn työkaluna

Ketterä ohjelmistokehitys yhdistää ohjelmistokehittämisen eri vaiheet iteraation sisälle. Iteraation suunnittelupalaverissa, kehitysjonosta nostetaan käyttäjätarinoita toteutukseen. Jokaisen iteraation aikana tehdään määrittelyä, suunnittelua, toteutusta ja testausta. Ketterässä ohjelmistokehityksessä prosessi aloitetaan vaatimusmäärittelystä, mutta alussa tehty vaatimusmäärittely ymmärretään suuntaa antavaksi ja määrittelyä tehdään alussa ensimmäisen iteraation tarvittava määrä eli sen verran, että sen perusteella kehittäminen saadaan käyntiin. (Luukkainen, 2020)

Sen sijaan, että yksityiskohtaiseen vaatimusmäärittelyyn käytetään paljon aikaa projektin alussa, ketterässä keskitytään toimittamaan asiakkaalle nopeasti ohjelmiston versioita iteraatioiden avulla. Nopea toimittaminen mahdollistaa jo etukäteen tehtyjen vaatimusten testaamisen ja validoiminen asiakkailla ja asiakkaiden edustajilla. Näin hyvin aikaisessa

vaiheessa projektia kehittävä tiimi, mahdolliset asiakkaan edustajat ja itse asiakas pystyvät varmistamaan, että vaatimukset ovat olleet oikeita ja ohjelmistoa kehitetään oikeaan suuntaan. Jokainen iteraatio toimittaa tuotantoon uusia toteutettuja vaatimuksia. Nämä toteutetut vaatimukset eli toiminnallisuudet eivät kuitenkaan ole välttämättä lopullisia, vaan ne kehittyvät iteraatioiden ja asiakkaiden palautteiden myötä. (Leffingwell, 2007, 2011; Luukkainen, 2020)

Iteraatioon valitaan toteutettavaksi ne vaatimukset eli käyttäjätarinat, joiden katsotaan tuovan asiakkaille eniten arvoa. Tavoite on, että asiakkaiden tarpeista nousseet vaatimukset saadaan nopeasti toteutettua ja ohjattua tuotantoon. Ohjelmiston valmistuessa pienissä osissa, asiakkailta on mahdollista saada palautetta ja näiden palautteiden avulla ohjelmiston vaatimuksia voidaan tarvittaessa täsmentää tai muuttaa. Ohjelmaversio ja siitä saatu palaute toimii syötteenä seuraavaan iteraatioon. (Auer et al., 2013; Luukkainen, 2020)

Jokainen valmis iteraatio mahdollistaa kehittävälle tiimille palautteen saannin ohjelmiston uudesta versiosta ja tätä kautta olemassa olevia vaatimuksia voidaan päivittää tai uusi lisätä sekä kehittämisen suuntaa hioa. Iteraatio onkin juuri tästä syystä tärkeä osa ketterän vaatimusmäärittelyä, koska asiakkaiden ymmärrys järjestelmästä kehittyy sitä mukaan, kun he pääsevät näkemään ja käyttämään ohjelmistoa. (Leffingwell, 2011)

5 Vaatimusmäärittely Digitaalinen viinikellari -sovellukselle

Opinnäytetyön käytännön osuudessa toteutettiin kehitystyönä vaatimusmäärittely uudelle ohjelmistolle. Kehitysprojekti toteutettiin hyödyntäen ketterän ohjelmistokehityksen arvoja, periaatteita ja työkaluja. Ketterän ohjelmistokehityksen työkaluista hyödynnettiin päivittäistä tilannepalaveria sekä Scrum-taulua, jolla töiden etenemistä seurattiin. Koska kyseessä oli yhden hengen projekti, projektimallia ei hyödynnetty käytännön työssä.

Opinnäytetyön teoriaosuudesta ja käytännön työstä syntyvää osaamista ja tietotaitoa hyödynnetään opinnäytetyön tekijän oman mielenkiinnon ja ammattitaidon kasvattamisessa. Opinnäytetyön kehitysprojektissa tehtyä uuden ohjelmiston vaatimusmäärittelyä ja siitä syntynyttä tuotteen kehitysjonoa tullaan hyödyntämään, kun ohjelmiston suunnittelua ja toteutusta jatketaan opinnäytetyön jälkeen.

Idea opinnäytetyön käytännön osuudessa hyödynnettävään ohjelmistoideaan nousi opinnäytetyön tekijän läpipiiristä. Viininharrastajat hankkivat, säilyttävät ja maistelevat erilaisia viinejä. Usein harrastajalla voi olla useita kymmeniä tai satoja viinejä omassa varastossaan. Usein harrastajalla on ostettujen viinipullojen tietoja varten jonkinlainen arkistointijärjestelmä, jonne merkitään muun muassa yksityiskohtia viinistä, lukumäärä sekä muita tietoja. Harrastajilla voi olla myös näiden lisäksi erilaisia muistiinpanoja esimerkiksi vihkoja tai Exceleitä, jonne he ovat merkitsevät maistelemiensa viinien arvosteluja ja omia kommenttejaan viineistä.

Uudessa ohjelmisto ideassa tavoite on luoda viininharrastajille sovellus, jonne voisi tallentaa tarvittavat tiedot ja tehdä hakuja näiden tietojen perusteella. Sen sijaan, että viiniharrastaja joutuu hallinnoimaan eri tietoja paperilla tai Excelissä, hän pystyisi tallentamaan tiedot yhteen sovellukseen. Tietojen löytyminen yhdestä paikasta voisi tarjota myös vielä tunnistamattomia synergiaetuja ja sekä erityisesti helpottaa tietojen hallintaa.

Opinnäytetyön käytännön osuuden kehitystyö aloitettiin vaatimusmäärittelyn ensimmäisestä osuudesta eli vaatimusten kartoittamisesta. Kun opinnäytetyön vaatimukset oli saatu kartoitettua, ne analysoitiin sekä dokumentointiin käyttäjätarinoina. Tämän jälkeen

vaatimukset priorisoitiin. Priorisoidut vaatimukset muodostavat tuotteelle kehitysjonon, jonka pohjalta sovelluksen suunnittelua ja toteuttamista jatketaan opinnäytetyön jälkeen.

5.1 Vaatimusten kartoittaminen

Digitaalinen viinikellari -sovelluksen vaatimusmäärittely aloitettiin vaatimusten kartoittamisella. Kyseessä on uusi ohjelmistoidea, jota ei ole työstetty etukäteen. Tämä korostaa vaatimusten kartoittamisen tärkeyttä, koska sovellus ei ole vielä käytössä ja tästä syystä esimerkiksi asiakaspalautteita ole mahdollista saada kehittämisen tueksi ja tukemaan kehittämisen suuntaa.

Ketterässä ohjelmistokehityksessä painotetaan välttämään etukäteen tehtävää määrittelyä ja suunnittelua, mutta tämä ei tarkoita, että määrittelyä tai suunnittelua ei tulisi tehdä ollenkaan etukäteen. Alkaessa kehittää varsinkin kokonaan uutta ohjelmistoa tulee ymmärtää mihin suuntaan kehittämisellä lähdetään. Ketterä ohjeistaakin, että määrittelyä ja suunnittelua tulee tehdä ainoastaan välttämätön määrä ja ymmärtää, että alussa tehty määrittely on vain suuntaa antavaa. Alussa tehtävään vaatimusmäärittelyyn ei myöskään tule käyttää huomattavaa määrää aikaa, koska todennäköisesti osa vaatimuksista tulee vielä muuttumaan tai päivittymään.

Ketterä ohjelmistokehitys antaa arvoja ja periaatteita, jotka katsotaan auttavat onnistumaan sovellusten kehittämisessä. Ketterä kuitenkin samaan aikaan korostaa, että tiimien tulee itse valita projektiin sopivat työkalut ja prosessit, kuitenkin niin että käytettävät välineet eivät ole ristiriidassa ketterän arvojen kanssa. Vaikka ketterä kannustaa välttämään etukäteen tehtävää määrittelyä ja suunnittelua, se ei kiellä sen tekemistä. Se korostaa kuitenkin, että tiimien tulee ymmärtää, että muutos ja epävarmuus ovat arkipäivää. Tiimin tulee miettiä miten paljon alussa tehtävään määrittelyyn kannattaa käyttää aikaa, vaatimukset tulevat kuitenkin muuttumaan. Toisaalta jos mitään vaatimustenmäärittelyä ei tehdä, investoidaanko turhan paljon itse ohjelmointiin, kun todennäköisesti iteraatioita tarvitaan enemmän, jos lähdetään etenemään täysin tyhjältä pöydältä.

Kehitysprojektin vaatimuksia kartoitettiin vertailemalla viiniharrastajille tarkoitettuja sovelluksia, asiakaskyselyllä (liite 2) sekä sovellusluonnoksella (liite 3), jonka avulla käyttäjä

pystyi tarkentaa näkemystään toiminnallisuuksista. Ei-toiminnalliset vaatimukset kartoitettiin tunnistamalla sovelluksen laadullisia vaatimuksia sekä rajoitteita. Ei-toiminnalliset vaatimukset ovat haastavia tunnistaa, mutta niiden tunnistaminen on tärkeää jo projektin alkuvaiheessa, jotta mahdolliset sovelluksen perustaan liittyvät vaatimukset pystytään ottamaan riittävän ajoissa huomioon.

5.1.1 Kilpailijavertailu

Viiniharrastajille tarkoitettujen sovellusten toiminnallisuuksia vertailemalla tavoiteltiin ymmärrystä siitä, minkälaisia vastaavia sovelluksia on olemassa ja minkälaisia toiminnallisuuksia niissä on. Kilpailijoiden kartoittaminen tehtiin tutustumalla eri sovelluskauppoihin ja etsimällä internetistä vastaavia sivustoja ja sovelluksia kuin Digitaalinen viinikellari -sovellus. Kilpailijavertailulla ei ollut tarkoitus kartoittaa markkinatilannetta vaan kartoittaa mahdollisia vaatimuksia sovellukselle. Sovelluksia testatessa keskityttiin sovellusten toimintojen tarkasteluun.

Kilpailevia sovelluksia etsiessä löydettiin muutamia vastaavia mobiilisovelluksia, mutta ei yhtään vastaavaa tai lähes vastaavaa selaimella käytettävää ohjelmaa. Suurin osa viiniharrastajille tarkoitetuista mobiilisovelluksista ja verkkosivuista vaikuttaa keskittyvät erilaisten viinien tietojen ja arvostelujen etsimiseen markkinoilta, ei omien tietojen tallentamiseen.

Vertailuun kohteeksi valittiin kaksi lähes vastaavaa mobiilisovellusta: My Wines ja MyWines - Wine Tracking App. Kummassakin sovelluksessa pystyy lisäämään omia viinejä ja hakemaan omien tallentamiensa viinien joukosta eri hakukriteereillä viinejä. Sovellusten löytämisen jälkeen sovelluksien toiminnallisuuksia tunnistettiin ja testattiin. Tunnistetut toiminnallisuudet tarjoavat vaatimusten kartoittamisvaiheessa ymmärrystä, siitä miten tunnistettuja tarpeita on ratkottu muissa sovelluksissa.

5.1.2 Asiakaskysely ja sovellusluonnos

Digitaalinen viinikellari – sovelluksen vaatimuksia kartoitettiin kilpailijavertailun lisäksi tekemällä sovelluksen kohderyhmälle asiakaskysely (liite 2) sovelluksen toiminnallisuuksista.

Asiakaskyselyn tavoite oli ymmärtää minkälaisia tarpeita ja odotuksia viiniharrastajilla on sovellukselle. Asiakaskysely lähetetiin kolmelle viininharrastajalle, jotka kaikki olivat harrastaneet viinejä jo yli 10 vuotta. Vastaajista kukaan ei hyödyntänyt mobiili – tai selainsovellusta tietojen tallentamiseen, vaan kaikilla oli käytössä paperinen arkistointijärjestelmä ja Excel. Vastaajat olivat 35–40-vuotiaita. Asiakaskyselyn toteutettiin pienelle vastaajaryhmälle, koska kyselyn tavoite oli saada kartoitettua tarpeellisia toiminnallisuuksia kevyesti. Asiakaskyselyn vastausaste oli 100 %.

Asiakaskyselyn lisäksi vaatimuksia kartoitettiin antamalla kohderyhmälle tutustuttavaksi sovellusluonnos (liite 3) ja pyytämällä käyttäjää tunnistamaan sovellusluonnoksen avulla toiminnallisuuksia ja halutessaan tarkentamaan asiakaskyselyyn kirjaamiaan toiminnallisuuksia. Sovellusluonnos toteutettiin Figma-sovelluksella. Sovellusluonnos ei ole suunniteltavan ohjelmiston prototyyppi, vaan se toteutettiin ainoastaan työvälineeksi vaatimusten kartoittamista varten. Tästä syystä sovellusluonnosta ei tarkastella yksityiskohtaisemmin tässä opinnäytetyössä. Vaikka sovellusluonnos on tehty mobiili käyttöliittymään, tämä ei tarkoita, että suunniteltava sovellus tullaan toteuttamaan ensin tai ollenkaan mobiiliin. Se toteutetaanko sovellus selaimeen vai esimerkiksi mobiiliin, riippuu asiakkaiden tarpeista.

Sovellusluonnos ja kommenttipyyntö lähetettiin asiakkaille kyselyn jälkeen, mutta luonnoksessa ei otettu huomioon kyselystä nousseita toiminnallisuuksia. Asiakasta pyydettiin arvioimaan luonnoksen toiminnallisuuksia ja miettimään mitä muita toiminnallisuuksia sovelluksessa olisi hyvä olla. Asiakasta ohjeistettiin, että sovellusluonnos ei ole kehitettävän ohjelmiston prototyyppi eli luonnoksen teksteihin ja kuviin ei kannata kiinnittää huomiota. Luonnon lähetettiin vasta asiakaskyselyn jälkeen, jotta luonnos ei vaikuttaisi käyttäjän vastauksiin asiakaskyselyssä. Asiakaskyselyn ja sovellusluonnoksen tavoite oli auttaa käyttäjiä hahmottamaan lisää toiminnallisuuksia ja tarkentamaan jo tunnistamiaan toiminnallisuuksia. Sovelluksen luonnos ja palautekysely lähetettiin samoille viininharrastajille kuin asiakaskyselykin oli lähetetty. Sovellusluonnoksen vastausaste oli 100 %.

Sovellusluonnosta haluttiin käyttää vaatimusten kartoittamiseen, koska koettiin että asiakkaat eivät välttämättä pysty tunnistamaan tarpeellisia toimintoja pelkän asiakaskyselyn

pohjalta. Jos kohderyhmä jolle asiakaskysely lähetettiin, olisi aikaisemmin käyttänyt vastaavaa sovellusta, toiminnallisuuksien tunnistaminen olisi ollut todennäköisesti helpompaa. Koska näin ei ollut koettiin, että sovellusluonnoksen hyödyntäminen vaatimusten kartoittamisessa on järkevää.

5.2 Vaatimusten analysointi

Kun vaatimukset on kartoitettu, ne analysoidaan. Kerätyistä vaatimuksista varmistettiin, että vaatimukset eivät ole ristiriidassa keskenään, vaatimukset ovat testattavissa ja että vaatimukset ovat järkevä toteuttaa. Kehitystyön tavoite ei ollut luoda tyhjentävää listaa yksityiskohtaisista vaatimuksista, koska tämä ei myöskään ole ketterän ohjelmistokehityksen arvojen ja periaatteiden mukaista, tavoite oli hahmotella mihin suuntaan kehittämistä kannattaa alkaa viemään.

Opinnäytetyön kehitysprojektissa vaatimusten analysoinnissa kaikki kartoitus vaiheessa tunnistetut ominaisuudet, toiminnallisuudet ja ei-toiminalliset vaatimukset kerättiin yhteen ja analysoitiin. Kilpailija vertailussa, asiakaskyselyssä ja sovellus luonnoksen kautta nousi samanlaisia vaatimuksia, nämä yhdistettiin. Kuten aikaisemmin todettiin, tavoite ei ollut tehdä tyhjentävää yksityiskohtaista vaatimuslistaa vaan luoda vaatimuksilla perusta, josta suunnittelu ja toteuttaminen voidaan aloittaa. Näin varmistetaan, että sovellus saadaan mahdollisimman nopeasti käyttäjien käytettäväksi ja käyttäjien antamaa palautetta päästään hyödyntämään kehitystyössä.

5.2.1 Kilpailijavertailu

Vertailuun valittiin kaksi lähes vastaavaa mobiilisovellusta: My Wines ja MyWines -Wine Tracking App. MyWines -Wine Trackin App vaikutti olevan suunniteltu Yhdysvaltojen markkinoille ja esimerkiksi viinin tietoihin tallennettavan hinnan valuutta ei voinut vaihtaa USD-dollarista muuksi valuutaksi. Sovellukset olivat toiminnallisuuksiltaan hyvin samanlaisia, kummankin sovelluksen kieli oli englanti ja kumpikaan sovellus ei vaatinut kirjautumista.

Hyviä toiminnallisuuksia MyWines -Wine Tracking App oli kuitenkin, että se tarjoaa mahdollisuuden tehdä varmuuskopion ja tallentaa itselleen kaikki sovellukseen syötetyt

viinien tiedot. Tämä on varmasti hyödyllinen toiminnallisuus, varsinkin jos käyttäjä on tallentanut sovellukseen paljon tietoa. Varmuuskopiointi on myös toiminnallisuus, jota käyttäjä ei välttämättä osaa nostaa itse esiin.

My Wines -sovelluksen idea oli yksinkertainen, vaikka siinä oli runsaasti eri toiminnallisuuksia. Kaikkien toiminnallisuuksien tarkoitusta ei ymmärretty sovellusta käyttäessä, mutta tämä taas liittyy vahvasti käyttöliittymän käytettävyyteen. Kaiken kaikkiaan My Wines -sovellus tarjosi mahdollisuuden tallentaa viinistä huomattavasti enemmän eri tietoja kuin MyWines -Wine Tracking App.

Kummassakin sovelluksessa on hyviä ominaisuuksia ja toiminnallisuuksia. Sovellusten läpikäyminen antoi hyvän näkökulman siihen, miten olemassa olevat sovellukset ovat pyrkineet ratkomaan opinnäytetyössäkin tunnistettuja viiniharrastajien tarpeita. Kilpailijavertailusta ei suoraan johdettu toiminnallisia tai ei-toiminnallisia vaatimuksia, vaan se tarjosi enemmänkin perspektiiviä asiakaskyselystä ja sovellusluonnoksesta nousseiden ominaisuuksien ja vaatimusten ymmärtämiseen.

5.2.2 Asiakaskysely ja sovellusluonnos

Asiakaskysely (liite 2) lähetettiin kolmelle viiniharrastajalle, jotka eivät ole aikaisemmin käyttäneet mitään sovellusta viinikellarinsa tietojen hallintaan. Vastauksissa nousi esiin tahtotila tallentaa sovellukseen viinien tietojen lisäksi viinien maistelujen tietoja ja tietoja, joita tarvitaan viinien maistelun järjestämiseen. Kyselyssä toivottiin myös, että viinien tietoja pystyisi tallentamaan hyvin laajasti ja halutessaan tiettyjä rivejä tai tietoja voisi piilottaa. Kyselyssä myös toivottiin, että sovelluksessa olisi tilaa omille muistiinpanoille, sekä mahdollisuus jonkinlaiselle omille tageille, joilla tallentamiaan viinin tietoja voisi yksilöidä niin kuin käyttäjä itse haluaisi.

Asiakaskysely oli kaiken kaikkiaan onnistunut. Kyselystä nousi esiin laajemmin tarpeita, kuin mitä oli ennen kyselyä tunnistettu. Vaikka vastaajien joukko oli pieni, vastauksissa nousi esiin yllättävän paljon erilaisia tarpeita. Itse kyselyn teosta opittiin myös paljon. Kun tehdään kysely asiakkaille tai mahdollisille käyttäjille, kyselyn kysymykset tulee olla hyvin tarkasti mietitty. Avoimet kysymykset kannustavat vastaamaan laajemmin, mutta riskinä on, että

kysymys on ymmärretty eri lailla, kuin kyselyn tekijä on kysymyksen tarkoittanut. Vaikka kysely lähetettiin pienelle ryhmälle, kyselyn teosta opittiinkin, että asiakaskyselyjen tekeminen on oma tieteenlajinsa, josta varmasti voisi myös kirjoittaa kokonaan oman opinnäytetyön.

Asiakaskyselyn lisäksi vaatimuksia kartoitettiin käyttämällä apuna sovellusluonnosta (liite 3). Sovellusluonnoksen palautteissa nousi esiin toiminnallisuuksien lisäksi, että käyttäjät toivoivat sovelluksesta yksinkertaista, selkeää ja käyttöä ohjaavaa. Tämä mukailee tämän päivän trendiä, jossa sovellusten käytettävyys on hyvin tärkeässä asemassa sovelluksen menestymisen näkökulmasta. Käyttäjät nostivat sovellusluonnoksen vapaamuotoisissa kommenteissa esiin tarpeen pystyä räätälöidä tallentamiaan tietoja omien tarpeidensa mukaisesti. Käyttäjät myös haluavat pystyä yhdistämään eri tallennettuja tietoja toisiinsa, esimerkiksi tallentamien viinien tietoja eri maisteluihin ja vastaavasti maistelujen muistiinpanoja viinien tietoihin. Kommenttien ja aikaisemmin saatujen asiakaskyselyjen yhteenvetojen kautta nousi esiin haaste, joka on tuttu varmasti monelle sovelluksen kehittäjälle: miten luoda sovellus, joka on samaan aikaan yksinkertainen ja selkeä, mutta mahdollistaa monen erilaisen tiedon tallentamisen ja räätälöinnin käyttäjän tahtotilan mukaan.

Asiakaskyselystä ja sovellusluonnoksesta nousseita tarpeita verrattiin kilpailevien sovellusten toiminnallisuuksiin. Näin saatiin ymmärrystä, miten joitakin tarpeita on ratkaistu muissa sovelluksissa. Asiakaskyselystä nousi esiin myös tarpeita, joita ei ole kilpailevissa sovelluksissa ollenkaan ratkottu. Asiakaskyselyyn peilatessa kilpailevat sovellukset ratkovatkin ainoastaan tarpeen tallentaa viinien tietoja, ei esimerkiksi tallentaa maistelujen tietoja ja muistiinpanoja.

Asiakaskyselystä ja luonnoksesta nousseet toiminnallisuudet ja ominaisuudet koostettiin yhteen ja niitä analysoitiin (taulukko 1). Vastauksista nostettiin esiin tarpeita, samanlaisuuksia ja varmistettiin että vaatimukset eivät ole ristiriidassa keskenään ja ne ovat testattavissa.

Toiminnallisten vaatimukset lisäksi asiakaskyselyn ja luonnoksen palautteiden pohjalta tunnistetut ei-toiminnalliset vaatimukset analysoitiin. Analysoinnissa huomattiin, että ei-

toiminnalliset vaatimukset vaativat vielä paljon tarkennusta. Ei-toiminnalliset vaatimukset kuitenkin tulevat tarkentumaan ohjelmiston suunnitteluvaiheen jälkeen, kun teknistä toteutusta suunnitellaan.

Taulukko 1 Yhteenveto analysoiduista vastauksista

Tiedot, jotka voi tallentaa ja joilla voi hakea		**Toiminnallisuudet / ominaisuudet**
Viinin perustiedot	**Viinin lisätiedot:**	Tallentaa tietoja
Rypäle	Kypsytysaika	Hakea tallennetuilla tiedoilla
Viinin nimi	Viinin tyyppi (jälkiruoka, rosee, puna, valkoviini, samppanja)	Oma profiili, jonka taakse tiedot tallentuvat
Viinitila/tuottaja	Hinta	Mahdollisuus käyttää sovellusta mobiilissa ja selaimessa
Tuotantovuosi	Mistä ja milloin hankittu	Mahdollisuus tehdä varmuuskopio
Tuotantomaa	Mahdollisuus lisätä erilaisia tageja tai itse yksilöityjä tietoja	Ilmoitus, kun viinin suositellun nauttimisajankohta on saavutettu
Tuotantoalue	Mahdollisuus lisätä linkkejä	Mahdollisuus tulostaa valitsemansa viinit listaksi
Alkoholimäärä	Mahdollisuus linkata maistatteluja itse viiniin (ja samalla maistattelun tiedot etc.)	Tulostaa pohja maisteluun, jossa näkyy valitut viinit ja jokaisen viinin kohdalla tilaa muistiinpanoille.
Sokerinmäärä	Tieto minkä ruokien kanssa sopii	Määritellä mitkä kentät näkyvät viinien tiedoissa
Viini kuvaus	**Varaston tiedot**	Määritellä omat hakukriteerit
Omat yleiset muistiinpanot	Varaston nimi	Tarjoaa osan viinin tiedoista (esim. viinialueen) suoraan kun lähtee kirjoittamaan
Oma profiili	Sijainti	Varaston kokonaiskuva: kuinka monta pulloa varastossa kokonaisuudessaan ja kuinka paljon käyttämätöntä tilaa
Maistatukset, joissa on käynyt	Kuinka monta pulloa mahtuu varastoon	Tietoturvallinen
Maistatuksien tiedot		Helppokäyttöinen
Mitä viinejä maisteltu		Looginen
Muistiinpanot		Saavutettava
Milloin pidetty ja missä		

5.3 Dokumentointi

Vaatimusten kartoittamisen ja analysoinnin jälkeen vaatimukset dokumentointiin. Usein luullaan, että ketterässä ohjelmistokehityksessä dokumentaatiota ei tehdä, mutta tämä on väärinymmärrys. Opinnäytetyön kehitysprojekti toteutettiin ketterän ohjelmistokehityksen arvoja ja periaatteita noudattaen, vaatimusmäärittelyssä ei hyödynnetty useille perinteisille projektimenetelmille tyypillistä vaatimusdokumenttia.

Toiminnalliset vaatimukset muutettiin käyttäjätarinoiksi ja dokumentoitiin. Ei-toiminnalliset vaatimukset dokumentointiin lauseina (taulukko 2). Vaatimusten dokumentoinnin yhteydessä vaatimuksia ei tarkennettu. Vaatimusten tarkentaminen tapahtuu, kun vaatimus otetaan iteraatiossa toteutettavaksi.

Taulukko 2 Dokumentoidut vaatimukset

Vaatimukset
Sovelluksen tulee täyttää EU:n tietosuoja-asetuksen (GDPR) vaatimukset
Sovelluksen tulee olla tietoturvallinen
Sovelluksen tulee pystyä toimimaan riittävän nopeasti, vaikka käyttäjä on tallentanut suuria määriä tietoja sovellukseen
Sovelluksen tulee olla saavutettava
Käyttäjänä haluan pystyä tallettamaan viinin tietoihin määriteltyjä lisätietoja, jotta pystyn halutessani tallettamaan kaikki tarpeelliset tiedot viineistäni
Käyttäjänä haluan pystyä liittämään maistelut omaan profiiliini, jotta pystyn tarkastelemaan missä kaikissa maisteluissa olen käynyt.
Käyttäjänä haluan pystyä luomaan itselleni profiilin, jotta pystyn kirjautumaan toisella laitteella tietoihini.
Käyttäjänä halua pystyä tallentamaan sovellukseen maisteluni tiedot esimerkiksi viinit, joita maisteltiin maistelussa ja muistiinpanoni, jotta kaikki maisteluni löytyvät yhdestä paikkaa.
Käyttäjänä haluan pystyä avaamaan viinin tiedot hakulistasta, jotta pystyn tarkastelemaan hakujani vastaavan viinin tietoja.
Käyttäjänä haluan pystyä muokkaamaan viinipullojen lukumäärää, jotta tiedän varaston ajantasaisen tilanteen, vaikka viinipullot vähenevät.
Käyttäjänä haluan nähdä viinipullojen määrän rypäleittäin ja varastoittain, jotta tiedän miten paljon minulla on mitäkin rypälettä varastossani.
Käyttäjänä haluan pystyä hakemaan viinin lisätiedoilla tallettamiani viinejä sovelluksesta, jotta löydän helposti tarpeisiini sopivan viinin.
Käyttäjänä haluan pystyä tekemään varmuuskopion kaikista sovellukseen tallentamistani tiedoista, jotta jos puhelimeni häviää tai sovellus hajoaa tiedot eivät häviä.
Käyttäjänä haluan pystyä hakemaan viinin perustiedoilla tallentamiani viinejä sovelluksesta, jotta löydän helposti tarpeisiini sopivan viinin.
Käyttäjänä haluan pystyä valitsemaan mitkä viinin tiedot näkyvät, kun viinin tiedot avataan, jotta pystyn näkemään vain minua kiinnostavat tiedot viinistä.
Käyttäjänä haluan pystyä valitsemaan, että sovellus tekee varmuuskopion automaattisesti, jotta minun ei tarvitse muistaa tehdä sitä aina kun tietoja muuttuu
Käyttäjänä haluan pystyä kirjatumaan omaan profiiliini, jotta pääsen näkemään omat tallentamiani tiedot.
Käyttäjänä haluan pystyä merkitsemään tiedon millä viinipullo löytyy varastosta, jotta pystyn löytämään haluamani viinipullon useiden viinipullojen joukosta helposti.
Käyttäjänä haluan pystyä tulostamaan listan valitsemistani viinistä, jotta pystyn välittämään haluamani viinien tiedot esimerkiksi ystävälleni.
Käyttäjänä haluan nähdä kuinka monta pulloa varastossani on ja kuinka monta pulloa sinne vielä mahtuu, jotta tiedän varastoni tilanteen.
Käyttäjänä haluan pystyä tallentamaan viinin perustiedot sovellukseen, jotta tiedän mitä viinejä minulla on varastossa
Käyttäjänä haluan pystyä kirjoittamaan omia muistiinpanojani viinin tietojen yhteyteen, jotta pystyn lisäämään haluamani tietoja.
Käyttäjänä haluan, että sovellus antaa minulle valmiita vaihtoehtoja mistä valita, kun tallennan viinin tietoja, jotta minun ei tarvitse kirjoittaa tietoa kokonaan
Käyttäjänä haluan merkitä missä varastossa viinipullo sijaitsee, jotta pystyn tarvittaessa löytämään sen.
Käyttäjänä haluan pystyä tallettamaan viinipullojen lukumäärän viinin tietoihin, jotta tiedän kuinka monta pulloa tiettyä viiniä varastossa on.
Käyttäjänä haluan pystyä hakemaan viinejä yhdellä tai useammalla tagilla, jotta pystyn löytämään sopivan viinin varastostani
Käyttäjänä haluan pystyä lisäämään viinin tietoihin tageja, jotta pystyn yksilöimään viinejä haluamillani tiedoilla
Käyttäjänä haluan pystyä tallentamaan, kuinka monta pulloa varastooni mahtuu kokonaisuudessaan, jotta tiedän miten iso varastoni on.
Käyttäjänä haluan tietää, jos olen maistelut viiniä jossakin maistelussa, jotta pystyn halutessani löytämään maistelussa tekemäni muistiinpanot kyseisen viinin osalta.

5.4 Validointi ja priorisointi

Viimeiseksi vaatimukset validoitiin eli ne käytiin läpi ja varmistettiin että ne ovat johdonmukaisia, ymmärrettäviä ja testattavissa. Vaatimuksista myös varmistettiin, että ne sopivat suunniteltavan ohjelmiston viitekehykseen.

Opinnäytetyön kehitysprojektin vaatimusten kartoittamisen, analysoinnin, dokumentoinnin ja validoinnin jälkeen vaatimukset priorisoitiin. Vaatimukset priorisoitiin asiakas arvon perusteella eli sen perusteella mikä toiminnallisuus tuottaisi suurinta lisäarvoa asiakkaalle ensimmäisenä. Arvioinnissa käytettiin hyväksi asiakaskyselyn vastauksia ja tarkasteltiin, onko toiminnallisuus noussut esiin useammalta vastaajalta. Arvioinnissa ja priorisoinnissa käytettiin hyväksi myös opinnäytetyön kirjoittajan ammattitaitoa arvioidessa mikä toiminnallisuus toisi asiakkaille eniten lisäarvoa.

Ei-toiminnalliset vaatimukset priorisoitiin kehitysjonon kärkeen. Ei-toiminnalliset vaatimukset vaativat kuitenkin huomiota koko kehitysprojektin ajan. Ei-toiminnalliset vaatimukset haluttiin priorisoida korkealle, koska ne tunnistettiin olevan ominaisuuksia, jotka tulee suunnitella ja mahdollistaa heti alusta lähtien. Ei-toiminnalliset vaatimukset eivät kuitenkaan häviä kehityslistalta, toisin kuin toiminnalliset vaatimukset, jotka on kuvattu käyttäjätarinoina.

Opinnäytetyön kehitysprojektin lopputuloksena syntyi tuotteen kehitysjono (liite 4). Kehitysjono koostuu tunnistetuista ja validoiduista vaatimuksista, jonka on priorisoitu. Kehitysjono painottuu hyvin vahvasti toiminnallisiin vaatimuksiin, mutta myös ei-toiminnallisia vaatimuksia on otettu huomioon. Ei-toiminnalliset vaatimukset kuten sovelluksen käytettävyys tulee tarkentumaan, kun tulevaisuudessa projektin kehittäminen jatkuu.

6 Johtopäätökset

Opinnäytetyön käytännön työssä toteutettiin Digitaaliselle viinikellari -sovellukselle vaatimusmäärittely ketterän ohjelmistokehityksen arvoja, periaatteita ja työkaluja noudattaen. Opinnäytetyön tietoperustassa tarkasteltiin vaatimusmäärittelyä osana ohjelmistokehittämistä ja sen roolia ketterässä ohjelmistokehityksessä.

Vaatimusmäärittelyn rooli ohjelmistokehityksessä on huomattava. Vaatimusmäärittelyllä varmistetaan, että kehitettävä ohjelmisto vastaa niihin asiakkaiden tarpeisiin, joihin sen on tarkoitus vastata. Ilman hyödyllisiä toimintoja, joita käyttäjät tarvitsevat, ohjelmisto on hyödytön.

Ohjelmistokehityksen historiassa näkyy vaatimusmäärittelyn tärkeä rooli ja sen mukanaan tuomat haasteet. Perinteisissä projektimalleissa vaatimukset lyödään lukkoon aivan projektin alussa, joka vaikeuttaa muuttuviin ja uusiin vaatimuksiin reagoimista huomattavasti. Tämä pahimmillaan aiheuttaa sen, että ohjelmisto on vanha jo valmistuessaan, eikä vastaa niihin tarpeisiin joihin sen oli tarkoitus vastata.

Ketterä ohjelmistokehitys pyrkii ratkaisemaan vaatimusmäärittelyssä tunnistetut haasteet muuttamalla kokonaan tapaa, jolla vaatimusmäärittelyä tehdään. Ketterässä ohjelmistokehityksessä vaatimusmäärittely muuttuu joustavaksi ja vaatimukset tarkentuvat, muodostuvat ja muuttuvat iteraatioiden myötä. Vaatimusmäärittelyä tehdään alussa vain sen verran, että suunta johon lähdetään saadaan selville. Tämän jälkeen iteraatioiden myötä valmistuu uusia ohjelmaversioita, joita asiakkaat ja asiakkaiden edustajat pääsevät käyttämään. Käytön perusteella syntyy palautetta ja mahdollisesti uusia vaatimuksia kehittämisen tueksi. Ketterässä ohjelmistokehityksessä vaatimusmäärittely poikkeaa siis perinteisestä vaatimusmäärittelystä huomattavasti, se on joustavaa ja iteraatioiden myötä rakentuvaa.

Vaikka perinteinen ja ketterä tapa tehdä vaatimusmäärittelyä poikkeavat toisistaan merkittävästi, kumpaankin tapaa hyödynnetään nykypäivänä. Kummassakin tavassa on tunnistetut haasteet ja hyvät puolet. Jos kehitettävän ohjelmiston vaatimukset ovat tiedossa kokonaisuudessa heti projektin alussa, esimerkiksi kokonaan lakiin perustuvissa projekteissa,

perinteinen tapa tehdä vaatimusmäärittelyä voi vastata tarpeeseen. Jos taas näin ei ole, ketterän tapa tehdä vaatimusmäärittelyä mahdollistaa asiakaslähtöisen ohjelmiston kehittämisen.

Ketterän ohjelmistokehityksen arvojen ja periaatteiden hyödyntäminen Digitaalinen viinikellari -sovelluksen vaatimusmäärittelyssä tuntui haastavalta, ei siksi, että ketterän hyödyntäminen yhden hengen projektissa on haastavaa, vaan siksi että ketterässä ohjelmistokehityksessä ei ole yksiselitteisiä ohjeita ja työkaluja miten vaatimusmäärittely tulisi tehdä. Ketterä ohjelmistokehitys tarjoaa arvoja, periaatteita, välineitä ja raameja, joita hyödyntäessä tekeminen on kevyempää ja tätä kautta ketterämpää. Ketterässä kuitenkin jätetään paljon tiimin omalle vastuulle. Ketterä korostaa, että tiimien tulee valita tilanteisiinsa ja itselleen parhaiten sopivat tavat ja välineet toimia. Kaiken tekemisen ja valintojen taustalla tulee kuitenkin aina olla ensisijaisena tavoitteena tuottaa asiakkaille heidän tarpeitansa täyttäviä versioita ohjelmistosta aikaisessa vaiheessa kehittämistä ja säännöllisesti kehittämisen edetessä. Näiden ajatusten pohjalta melkein voisi sanoa, että ketterässä ohjelmistokehityksessä ei ole yhtä täysin oikeaa tapaa tehdä vaatimusmäärittelyä, niin kauan kun tekeminen noudattaa ketterän ohjelmistokehityksen arvoja ja periaatteita sekä mahdollistaa asiakkaille heidän tarpeitansa täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti, se on ketterää.

Opinnäytetyön käytännön työtä koskeva tutkimuskysymys oli: Sopiiko ketterän ohjelmistokehityksen tapa tehdä vaatimusmäärittelyä pieneen projektiin? Vastaus kysymykseen on, että kyllä sopii. Ketterän ohjelmistokehityksen tapaa tehdä vaatimusmäärittelyä sopii myös pieniin projekteihin, kunhan valitsee projektin ja sen etenemisen kannalta sopivat työvälineet pitäen mielessä ketterä ohjelmistokehityksen arvot ja periaatteet. Pienet projektit ovat varmasti jo lähtökohtaisesti ketteriä, mutta myös pienissä projekteissa järjestelmällisyys on tärkeää. Tämän lisäksi ensisijaisesti tulisi aina muistaa, että ohjelmia tehdään käyttäjille. Vaikka kuinka houkuttaisi lähteä koodaamaan suoraan, aina on järkevä ensin pysähtyä miettimään mitä halutaan todellisuudessa saavuttaa ja mitkä vaatimukset meitä ohjaa.

Opinnäytetyön kehitysprojekti osoitti, että vaikka kyseessä olisi pieni sovellus, vaatimusmäärittelyn rooli on tärkeä ja asiakkaiden osallistuttaminen vaatimusten

kartoittamiseen on ehdotonta. Näin varmistetaan että kehitettävät toiminnallisuudet perustuvat aidosti käyttäjien tarpeisiin, ei vain luuloon käyttäjinen tarpeista. Vaikka ymmärrys asiakkaiden todellisista tarpeista on tärkeä, tarkoitus ei kuitenkaan ole kartoittaa yksityikohtaisia tarpeita etukäteen ja lähteä päättäväisesti toteuttamaan vain alussa kartoitettuja vaatimuksia. Maailma muuttuu ja asiakkaiden tarpeet sekä mieltymykset voivat muuttua nopeastikin. Toiseksi asiakas ei välttämättä aina tiedä mitä hän oikeasti haluaa. Ketterä ohjelmistokehityksen arvot ja periaatteet pureutuvatkin juuri tähän, ajatukseen, että muutoksia tulee aina. Tästä syytä suuurin ja tärkein vaatimusmäärittely ketterässä tehdään iteraatioissa, tarkentamalla vaatimukset juuri ennen toteutusta sekä reagoimalla palautteeseen ja tarkentuneisiin vaatimuksiin, jotka ovat syntyneet, kun asiakas on päässyt käyttämään sovelluksen versiota.

Vaikka opinnäytetyön kehitysprojektissa oli haasteita vaatimusmäärittelyn kanssa, työ oli onnistunut. Opinnäytetyöstä saatiin valtavasti ymmärrystä ja käytännön työ tarjosi myös kokemusta teorian tueksi. Tämän lisäksi jos Digitaalista viinikellari -sovellusta olisi lähdetty ohjelmoimaan suoraan, ilman minkäänlaista vaatimusmäärittelyä alussa, kehittämisen suunta olisi todennäköisesti ollut erilainen. Tämä tarkoittaa sitä, että sovellus olisi todennäköisesti tarvinnut useampia iteraatioita, ennen kuin se olisi saavuttanut saman pisteen kuin mistä kehittäminen nyt lähtee liikeelle. Tämä oppi ei kuitenkaan tarkoita sitä, että tuotteen Digitaalinen viinikellari -sovelluksen kehitysjono lyödään lukkoon opinnäytetyössä tehdyn vaatimusmäärittelyn pohjalta, vaan tämä tarkoittaa sitä että kehittämisen suunta eli pohja josta sovelluksen kehittämistä tullaan jatkamaan, on nyt selkeämpi. Toisin kuin perinteisessä vaatimusmäärittelyssä, ketterän vaatimusmäärittelyn perusteella ei niinkään synny majakkaa, jota kohti lähdetään kulkemaan vaan enemmänkin pohja, jolta lähdettään ponnistamaan.

7 Yhteenveto

Opinnäytetyön tavoitteena oli ymmärtää mitä vaatimusmäärittely on, mikä rooli vaatimusmäärittelyllä on ketterässä ohjelmistokehityksessä ja soveltuuko ketterän tapa tehdä vaatimusmäärittelyä pieniin projekteihin. Opinnäytetyön tukimuskysymyksiin vastaaminen onnistui hyvin. Ymmärrys vaatimusmäärittelyn tarkoituksesta ja roolista selkeni itselleni opinnäytetyön myötä. Ketterä ohjelmistokehitys suhtautuu vaatimusmäärittelyyn erilailla kuin perinteiset projektimallit, mutta vaatimusmäärittelyllä on todella tärkeä rooli myös ketterässä ohjelmistokehityksessä.

Opinäytetyön käytännön projektissa toteutettiin vaatimusmäärittely uudelle ohjelmistolle. Samassa yhteydessä, käytännön työn kautta, takasteltiin sopiiko ketterän ohjelmistokehityksen tapa tehdä vaatiusmäärittelyä myös pieneen projektiin. Käytännön työn kautta vaatimusmäärittelyn rooli konkretisoitui itselleni. Kyseessä ei ole pieni, eikä helppo kokonaisuus, mutta mielestäni ketterän arvot, periaatteet, työkalut ja projektimallit tukevat ja ohjaavat tekemistä oikeaan suuntaan.

Kaiken kaikkiaan opinnäytetyö onnistui mielestäni hyvin ja tavoiteltu osaamisen kasvattaminen toteutui. Nyt minulla on huomattavasti vahvempi ymmärrys ja osaaminen vaatimusmäärittelystä kuin ennen opinnäytetyön tekemistä. Opinnäytetyön pohjalta itselleni on myös selkeytynyt ymmärrys, että kaikki projektiin osallistuvien, myös teknistä toteutusta tekevien, tulee ymmärtää käytettävän projektimallin tapa toimia, sen tavoitteet, oma rooli projektissa sekä kehitystyön elinkaaret ja varsinkin vaatimusmäärittelyn rooli. Näin pystytään varmistamaan, että jokainen osallistuja ymmärtää miten projektia ollaan edistämässä ja mitä itseltä odotetaan.

Opinnäytetyön kehitysprojektissa tehtiin vaatimusmäärittely Digitaalinen viinikellari -sovellukselle. Opinnäytetyöstä konkreettisesti valmistui tuotteelle priorisoitu kehitysjono, josta kehittämistä tullaan jatkamaan. Opinnäytetyön jälkeen Digitaalinen viinikellari -sovelluksen tarkempi suunnittelu ja toteutus käynnistyy. Tämän lisäksi opinnäytetyön teoriaosuudessa opittuja asioita tullaan hyödyntämään työelämässä.

Lähteet

Agile Alliance. (2001a). *Agile Manifesto for Software Development | Agile Alliance*. https://www.agilealliance.org/agile101/the-agile-manifesto/

Agile Alliance. (2001b). *Julistuksen takana olevat periaatteet*. https://agilemanifesto.org/iso/fi/principles.html

Agile Alliance. (2001c). *Ketterän ohjelmistokehityksen julistus*. https://agilemanifesto.org/iso/fi/manifesto.html

Agile Alliance. (2021). *What is Agile Software Development? | Agile Alliance*. https://www.agilealliance.org/agile101/

Auer, A., Auer, L., Heinäsmäki, M., Hölttä, J., Kalliala, E., Laanti, M., Laine, K., Lekman, L., Miinalainen, P., Naski, H., Piiparinen, T., Puhakka, H., Pyhäjärvi, M., Pääkkönen, T., Räisänen, S., Sora, H., Taipale, M., Talvio, J., Tanninen, A., … von Weissenberg, M. (2013). *Ketterää kehitystä*. Finn Lectura.

Behutiye, W., Karhapää, P., Costal, D., Oivo, M., & Franch, X. (2017). Non-functional requirements documentation in agile software development: Challenges and solution proposal. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10611 LNCS*, 515–522. https://doi.org/10.1007/978-3-319-69926-4_41

Digital.ai. (n.d.). *14th annual State of Agile Report | State of Agile*. Retrieved June 23, 2021, from https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494

Haikala, I., & Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt*. Telentum.

Jokela, T. (2009). *Toivomuslistoista todennettavaan käytettävyyteen*. http://www.pcuf.fi/sytyke/lehti/kirj/st20093/ST093-18A.pdf

Juvonen, R. (2018). *Ohjelmistoprojektin sudenkuopat ja miten ne vältetään*. Books on Demand.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for a Large Enterprises*. Addison-Wesley.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise*. Addison-Wesley.

Leppänen, T. (2021). *Mihin tarvitaan ei-toiminnallisia IT-vaatimuksia?* https://www.cheetah.fi/blog/mihin-tarvitaan-ei-toiminnallisia-it-vaatimuksia/

Luukkainen, M. (2010). *Ohjelmistojen mallintaminen, Johdatus ohjelmistotuotantoon*. https://www.cs.helsinki.fi/u/pohjalai/ke10/ohma/slides/ohma_01-ohjelmistotuotannosta.pdf

Luukkainen, M. (2020). *Ohjelmistotuotanto 2020*. https://ohjelmistotuotanto-hy.github.io/

Ojasalo, K., Moilanen, T., & Ritalahti, J. (2020). *Kehittämistyön menetelmät - Uudenlaista osaamista liiketoimintaan*. Sanoma Pro Oy.

Poimala, S., & Tolvanen, P. (n.d.). *Ketteryys haltuun: Scrum pähkinänkuoressa – Sininen Meteoriitti*. Retrieved June 17, 2021, from https://meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/

Royce, W. W. (1970). *Managing the development of large software systems*. https://blog.jbrains.ca/assets/articles/royce1970.pdf

Saddington, P. (2012). *The Agile Pocket Guide : A Quick Start to Making Your Business Agile Using Scrum and Beyond*. John Wiley & Sons. https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/detail.action?docID=947659.

Scrum.org. (2021). *Professional Scrum Competency: Understanding and Applying the Scrum Framework | Scrum.org*. https://www.scrum.org/professional-scrum-competencies/understanding-and-applying-scrum-framework

Taina, J. (2009). *Ohjelmistoprosessit ja ohjelmistojen laatu*. https://www.cs.helsinki.fi/u/taina/opol/k-2009/pdf/luku-6_2.pdf

Toikko, T., & Rantanen, T. (2009). *Tutkimuksellinen kehittämistoiminta*. Tampereen Yliopistopaino Oy.

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojekti:

Opinnäytetyön kehitysprojektia varten tehdään kysely asiakkaille (liite 2) ja pyydetään palautetta sovellusluonnoksesta (liite 3). Kyselyn vastaukset ja sovellusluonnoksen palautteet analysoidaan ja hyödynnetään kehitysprojektissa. Kyselyjen yhteydessä pyydetään kyselyyn vastaajalta kirjallinen lupa annettujen vastausten hyödyntämiseen opinnäytetyössä. Palaute sovellusluonnoksesta pyydetään vapaamuotoisena ja palautteen antajaa ohjeistetaan, että antamalla palautetta asiakas hyväksyy, että analysoitua palautetta hyödynnetään opinnäytetyössä. Vastaajien sähköpostiosoitetta tai mitään henkilötietoja ei tallenneta missään vaiheessa. Kehitysprojektin aikana pidetään myös päiväkirjaa, johon kerätään tietoja projektin etenemisestä.

Kaikkia opinnäytetyön kehitysprojektissa syntyneitä aineistoja ja opinnäytetyön versioita säilytetään tekijän tietokoneen C-asemalla, josta tehdään säännöllisesti työstön aikana varmuuskopioita ulkoiselle kovalevylle. Kehitystyöprojektin materiaaleja säilytetään ulkoisella kovalevyllä vähintään yhden vuoden opinnäytetyön valmistumisesta. Tämän jälkeen aineistot tuhotaan.

Opinnäytetyön tekijä omistaa opinnäytetyön kehitysprojektin aineistot ja tulokset.

Liite 2: Asiakaskysely: Digitaalinen viinikellari –sovelluksen toiminnallisuudet

Kyselyn tarkoituksena on kartoittaa Digitaalinen viinikellari -sovelluksen toiminnallisuuksia. Kyselyssä pääset kertomaan mielipiteesi sovelluksen toiminnallisuuksista sekä miten sinä haluaisit, että Digitaalinen viinikellari –sovellus toimisi. Kysely on osa opinnäytetyötä, jossa kartoitetaan Digitaalinen viinikellari –sovelluksen vaatimuksia.

☐ Suostun, että vastauksiani hyödynnetään Sanna Liljan opinnäytetyössä.
Kyselyn vastaukset käsitellään nimettöminä ja vastaajan sähköpostiosoitetta tai henkilötietoja ei tallenneta.

Mitä tietoja haluaisit pystyä tallentamaan sovellukseen?

Valitse 3–4 tärkeintä tietoa mitkä koet, että tulisi pystyä tallentamaan sovellukseen? Perustele valintasi.

1.
2.
3.
4.

Millä tiedoilla haluaisit etsiä sovelluksesta tietoja?

Valitse 3–4 tärkeintä tietoa, joilla tulisi pystyä hakemaan sovelluksesta? Perustele valintasi.

1.
2.
3.
4.

Mitä muita toimintoja haluaisit, että sovelluksella pystyy tekemään? Perustele vastauksesi.

Käyttäisitkö ohjelmaa ensisijaisesti

☐ Internet- selaimella

☐ Mobiilisovelluksella

Koetko, että Digitaalinen viinikellari –sovellus voisi olla hyödyllinen sinulle? Perustele vastauksesi.

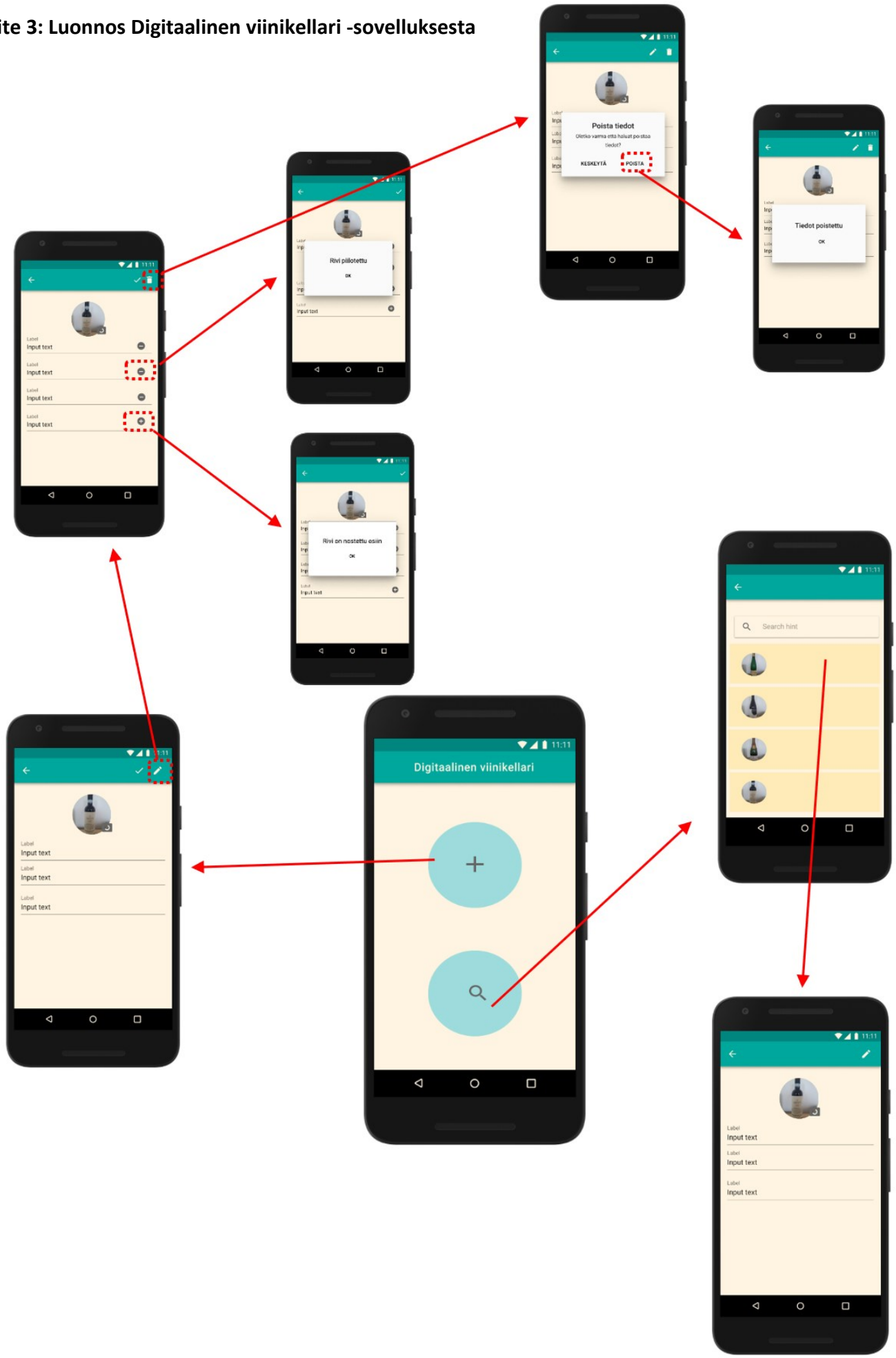
Missä tilanteissa ensisijaisesti käyttäisit Digitaalinen viinikellari – sovellusta? esimerkiksi tallentaessasi tietoja uudesta viinistä, viinin maistelun muistiinpanojen tallentamiseen, etsiessäsi sopivaa viiniä nautittavaksi tai jossakin muussa tilanteessa, missä?

Kiitos vastauksistasi!

Vastaukset kyselyyn voit palauttaa osoitteeseen: sanna.ka.lilja@student.hamk.fi

Mahdollisissa kyselyyn liittyvissä kysymyksissä voi olla yhteydessä yllä olevaan sähköpostiin.

Liite 3: Luonnos Digitaalinen viinikellari -sovelluksesta



Liite 4: Digitaalinen viinikellari –sovelluksen kehitysjono

Digitaalinen viinikellari -sovelluksen kehitysjono	prioriteetti
Sovelluksen tulee täyttää EU:n tietosuoja-asetuksen (GDPR) vaatimukset	1
Sovelluksen tulee olla tietoturvallinen	2
Sovelluksen tulee pystyä toimimaan riittävän nopeasti, vaikka käyttäjä on tallentanut suuria määriä tietoja sovellukseen	3
Sovelluksen tulee olla saavutettava	4
Käyttäjänä haluan pystyä luomaan itselleni profiilin, jotta pystyn kirjautumaan toisella laitteella tietoihini.	5
Käyttäjänä haluan pystyä kirjatumaan omaan profiiliini, jotta pääsen näkemään omat tallentamani tiedot.	6
Käyttäjänä haluan pystyä tallentamaan viinin perustiedot sovellukseen, jotta tiedän mitä viinejä minulla on varastossa	7
Käyttäjänä haluan pystyä hakemaan viinin perustiedoilla tallentamiani viinejä sovelluksesta, jotta löydän helposti tarpeisiini sopivan viinin.	8
Käyttäjänä haluan pystyä avaamaan viinin tiedot hakulistasta, jotta pystyn tarkastelemaan hakujani vastaavan viinin tietoja.	9
Käyttäjänä haluan pystyä tallettamaan viinipullojen lukumäärän viinin tietoihin, jotta tiedän kuinka monta pulloa tiettyä viiniä varastossa on.	10
Käyttäjänä haluan pystyä muokkaamaan viinipullojen lukumäärää, jotta tiedän varaston ajantasaisen tilanteen, vaikka viinipullot vähenee.	11
Käyttäjänä haluan pystyä tallettamaan viinin tietoihin määriteltyjä lisätietoja, jotta pystyn halutessani tallettamaan kaikki tarpeelliset tiedot viineistäni	12
Käyttäjänä haluan pystyä hakemaan viinin lisätiedoilla tallettamiani viinejä sovelluksesta, jotta löydän helposti tarpeisiini sopivan viinin.	13
Käyttäjänä haluan pystyä tekemään varmuuskopion kaikista sovellukseen tallentamistani tiedoista, jotta jos puhelimeni häviää tai sovellus hajoaa tiedot eivät häviä.	14
Käyttäjänä haluan pystyä lisäämään viinin tietoihin tageja, jotta pystyn yksilöimään viinejä haluamillani tiedoilla	15
Käyttäjänä haluan pystyä hakemaan viinejä yhdellä tai useammalla tagilla, jotta pystyn löytämään sopivan viinin varastostani	16
Käyttäjänä haluan pystyä kirjoittamaan omia muistiinpanojani viinin tietojen yhteyteen, jotta pystyn lisäämään haluamani tietoja.	17
Käyttäjänä haluan, että sovellus antaa minulle valmiita vaihtoehtoja mistä valita, kun tallennan viinin tietoja, jotta minun ei tarvitse kirjoittaa tietoa kokonaan	18
Käyttäjänä haluan pystyä valitsemaan mitkä viinin tiedot näkyvät, kun viinin tiedot avataan, jotta pystyn näkemään vain minua kiinnostavat tiedot viinistä.	19
Käyttäjänä haluan pystyä valitsemaan, että sovellus tekee varmuuskopion automaattisesti, jotta minun ei tarvitse muistaa tehdä sitä aina kun tietoja muuttuu	20
Käyttäjänä haluan merkitä missä varastossa viinipullo sijaitsee, jotta pystyn tarvittaessa löytämään sen.	21
Käyttäjänä haluan pystyä merkitsemään tiedon millä viinipullo löytyy varastosta, jotta pystyn löytämään haluamani viinipullon useiden viinipullojen joukosta helposti.	22
Käyttäjänä haluan pystyä tallentamaan kuinka monta pulloa varastooni mahtuu kokonaisuudessaan, jotta tiedän miten iso varastoni on,	23
Käyttäjänä halua pystyä tallentamaan sovellukseen maisteluni tiedot, viinit, joita maisteltiin maistelussa ja muistiinpanoni, jotta kaikki maisteluni löytyvät yhdestä paikkaa.	24
Käyttäjänä haluan pystyä liittämään maistelut omaan profiiliini, jotta pystyn tarkastelemaan missä kaikissa maisteluissa olen käynyt.	25
Käyttäjänä haluan tietää, jos olen maistelut viiniä jossakin maistelussa, jotta pystyn halutessani löytämään maistelussa tekemäni muistiinpanot kyseisen viinin osalta.	26
Käyttäjänä haluan nähdä viinipullojen määrän rypäleittäin ja varastoittain, jotta tiedän miten paljon minulla on mitäkin rypälettä varastossani.	27
Käyttäjänä haluan nähdä kuinka monta pulloa varastossani on ja kuinka monta pulloa sinne vielä mahtuu, jotta tiedän varastoni tilanteen.	28
Käyttäjänä haluan pystyä tulostamaan listan valitsemistani viinistä, jotta pystyn välittämään haluamani viinien tiedot esimerkiksi ystävälleni.	29