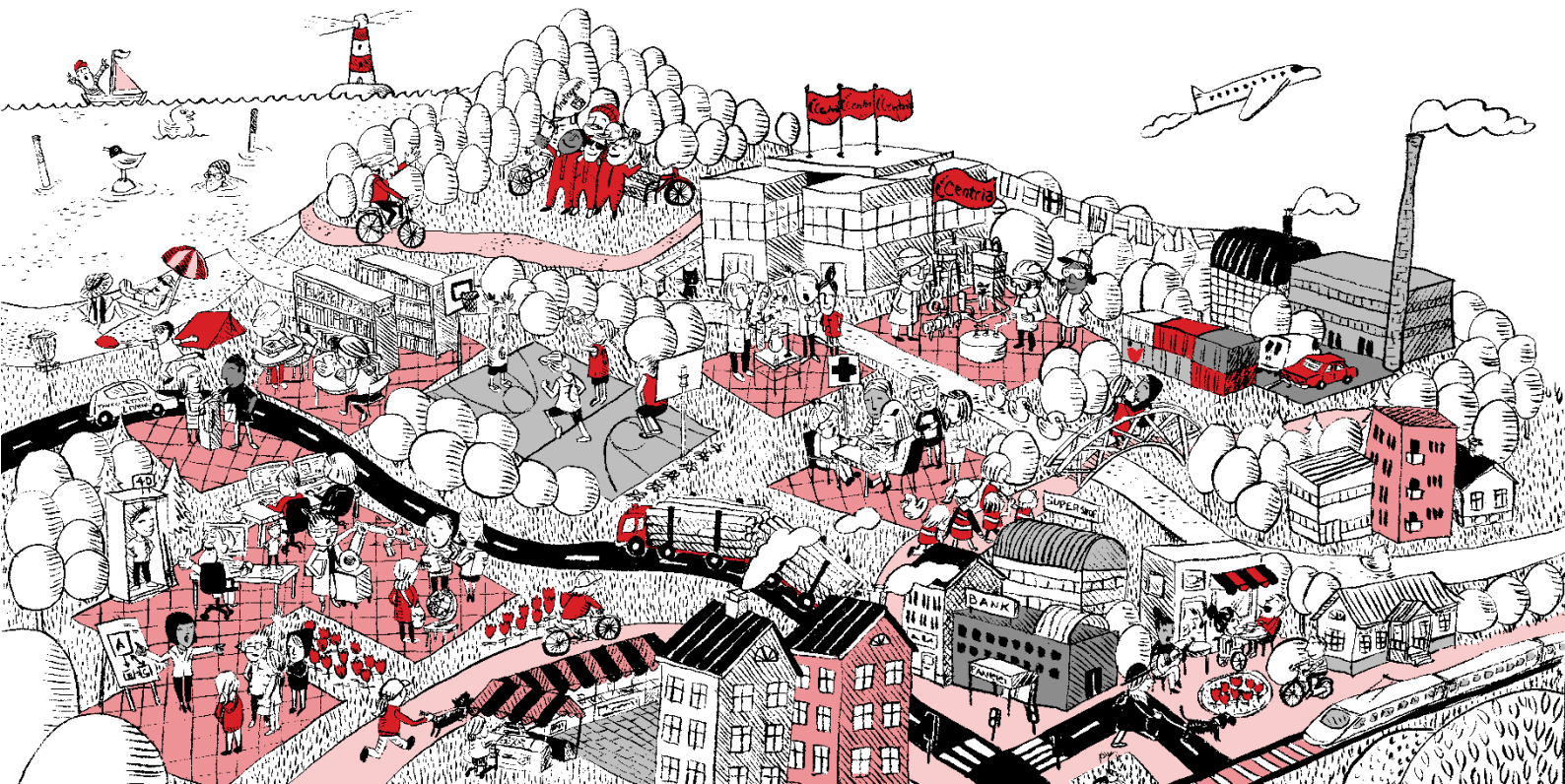
Tom Cygnel

OHJELMISTOPROTOTYYPIN KEHITTÄMINEN

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Digitalisaation johtamisen koulutusohjelma
Kevät 2022



TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Kevät 2022	**Tekijä/tekijät** Tom Cygnel
Koulutus Digitalisaation johtaminen		☐ AMK ☒ YAMK
Työn nimi Ohjelmistoprototyypin kehittäminen		
Työn ohjaaja Jari Isohanni		**Sivumäärä** 33+3
Työelämäohjaaja Miikka Puumala		

Työn tarkoituksena oli tehdä työkirja ohjelmiston käyttäjäkokemuksen, käyttöliittymän graafisen suunnittelun sekä prototyypin kehittämiselle.

Haasteena on monesti se, että ohjelmiston visuaalinen suunnitteluprosessi koostuu pelkästään graafisesta suunnitelmasta. Keksitään esimerkiksi hyvältä näyttävä etusivu tai tapa esittää tietoa raportissa. Tämä ei kuitenkaan ole riittävä, koska vaaditaan myös ohjelmiston kehittämisosaamista. Suunnitelma saattaa toimia ja näyttää hyvältä siihen käyttöön, johon se on tehty, mutta se kuitenkaan toimi oikeassa teknisessä ympäristössä.

Vaikka käyttäjäkokemussuunnittelu saattaa vaikuttaa raskaalta prosessilta, sen tarkoitus on kuitenkin säästää aikaa ja rahaa pidemmällä aikavälillä, tekemällä tuotteista mahdollisimman virheettömiä heti alkuun. Säästö tulee, kun virheiden korjaaminen alkuvaiheessa on paljon edullisempaa kuin korjaus tuotannossa olevaan tuotteeseen. Lisäksi monen tiimin jäsenen työ helpottuu ja nopeutuu, kun saa prototyypin aikaisen vaiheen käyttäjäpalautteista selkeän suunnan, jota kohti ollaan menossa.

Tässä työssä selvitettiin, miten Raisoftin ohjelmiston suunnitteluprosessi toimii ja miten käyttökokemus on otettu tällä hetkellä huomioon. Valittiin tiettyjä teorioita, jotka voisivat toimia hyvin Raisoftin tyypillisissä projekteissa, ja näiden pohjalta tehtiin prototyyppikehittämisen työkirja. Työkirjaa kokeiltiin käytännössä, ja tästä kerättiin tuloksia.

Tulokset kertoivat, että käyttäjätarinat olivat hyvä tapa kommunikoida tarpeista. Lisäksi nähtiin, että tarkempi dokumentaatio kannatti tehdä vasta sen jälkeen, kun prototyyppi oli testattu.

Asiasanat
Ohjelmistosuunnittelu, UX/UI, ohjelmisto, prototyyppi

ABSTRACT

Centria University of Applied Sciences	**Date** Spring 2022	**Author** Tom Cygnel
Degree programme **Digitalisaation johtaminen**		
Name of thesis Development of software prototypes		
Centria supervisor Jari Isohanni		**Pages** 33+3
Instructor representing commissioning institution or company Miikka Puumala		

The purpose of the work was to make a workbook for the development of the software user experience, user interface (UX/UI) graphic design and prototype.

The challenge is often that the visual design process of the software consists solely of a graphic design. For example, someone designed a good-looking front page or a way to present information in a report. However, this is not enough. You must also be able to use the software. The plan may work and look good for the situation in which it was made, but it will not work in the right way.

While user experience design may seem like a cumbersome process, it is designed to save time and money in the long run by making the product as flawless as possible from the start. The savings come from the fact that correcting errors at an early stage is cheaper than repairing a product production. In addition, the work of many team members is made easier and faster when you get a clear direction from the prototype's early-stage user feedback to where you are heading.

In this work, it was investigated how Raisoft's software design process works and how the user experience is currently considered. Certain theories that could work well in Raisoft's typical projects were selected, and a prototype development workbook was developed based on these. The workbook was tested in practice and results were collected.

The results showed that using “user stories” was a good way to communicate the needs. It was found that detailed specifications in the early stages of development were not beneficial. Instead, an early prototype gave batter results in collection feedback.

Key words
Software design, UX/UI, software prototypes

KÄSITTEIDEN MÄÄRITTELY

UX/UI

User experience / User interface - Käyttökokemus ja käyttöliittymä

RAIsoft.net

Raisoftin päätuote

Jira

Tehtävien hallintaan tehty kaupallinen ohjelmisto

Refaktorointi

Prosessi, jossa ohjelman lähdekoodi ja rakenne muutetaan tai uudistetaan, mutta toiminnallisuus säilyy.

SAAS

Software As A Service – Ohjelmisto palveluna

Adobe XD

Prototyyppikehitykseen kehitetty kaupallinen ohjelmisto

PowerPoint

Microsoftin tekemä ohjelmisto esitysten laatimiseen

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO

Laadukas ohjelmisto sisältää niitä ominaisuuksia, jotka täyttävät asiakkaan tarpeet ja sen kautta pitävät asiakaan tyytyväisenä. Laadukas tuote on myös sellainen, jota on helppo käyttää. Tyytyväinen asiakas on laadun näkökulmasta lopullinen tavoite. Jos ohjelmistossa esiintyy ongelmia julkaisun jälkeen, ovat korjaukset yleensä enemmän aikaa vieviä korjata, ja siten kalliimpia. Ongelmien ja puutteiden löytäminen aikaisessa vaiheessa tuo siis säästöä pitkällä aikavälillä.

Ohjelmiston suunnittelu lähtee Raisoftilla monesti graafisesta suunnittelusta, jolloin esimerkiksi keksitään idea hyvältä näyttävästä raportista tai ohjelmistomoduulista. Tällaisessa suunnittelussa ei välttämättä oteta tarkasti huomioon asiakkaiden todellisia tarpeita tai sitä, osaavatko käyttäjät käyttää ohjelmistoa. Loppukäyttäjät näkevät tuotteen yleensä ensimmäistä kertaa vasta siinä vaiheessa, kun se asennetaan heille testi- tai tuotantokäyttöön.

Tässä työssä tavoitteena on löytää menetelmiä, joilla voidaan parantaa ohjelmiston laatua luomalla asiakkaille arvokkaita ominaisuuksia. Raisoftilla on käytössä ennestään erilaisia menetelmiä, kuten esimerkiksi Scrum-prosessi. Työn tavoitteena on löytää uusia menetelmiä, joilla voidaan tehostaa ohjelmistotuotantoa entisestään.

Tehtävänä on luoda parempi prosessi käyttöliittymän suunnitteluun. Tavoitteena on suunnitella prosessi, jossa muun muassa prototyyppien avulla testataan ja varmistetaan, että ollaan ratkaisemassa asiakkaan oikeat ongelmat ja tehdään se tavalla, jolla ohjelmiston visuaalinen yhtenäisyys ja helppokäyttöisyys säilyvät. Tätä mallia testataan uuden ohjelmistomoduulin suunnittelussa.

2 TOIMINTAYMPÄRISTÖN KUVAUS

2.1 Raisoft

Raisoft on vuonna 2000 Kokkolassa perustettu ohjelmistoalan yritys. Yrityksen päätuote on Raisoft.net. Yrityksessä työskentelee noin 60 henkilöä, joista noin puolet työskentelee ohjelmistokehityksessä. Raisoftin asiakkaat Suomessa ovat kunnat ja yksityiset hoitokodit. Suurin osa Suomen kunnista käyttää Raisoft ohjelmistoja. Suomen lisäksi Raisoft on alansa markkinajohtaja myös Sveitsissä. Kehitysyhteistyötä on myös mm. Australiassa, Uudessa Seelannissa, Singaporessa ja Espanjassa. (Raisoft 2022.)

2.2 RAI-järjestelmä sekä RAIsoft

RAI-järjestelmä on kansainvälisen tutkijaorganisaation, interRAI-järjestön, luoma yksilön kokonaisvaltaisen toimintakyvyn arviointiväline. Arviointivälineet tulevat Suomeen Terveyden ja hyvinvoinnin laitoksen, THL:n kautta. Välineitä käytetään mm. vanhustenhuollossa, asiakasohjauksessa, ennaltaehkäisevässä työssä sekä mielenterveys- ja kehitysvammatyössä. (Raisoft 2022.)

RAIsoft-ohjelmistot on rakennettu interRAI-arviointivälineiden ympärille. RAI-järjestelmä on standardoitu tiedonkeruun ja havainnoinnin välineistö, joka on tarkoitettu vanhus- tai vammaispalvelun asiakkaan palvelutarpeen arviointiin sekä hoito-, kuntoutus- ja palvelusuunnitelman laatimiseen. Raisoft-ohjelman avulla tehty interRAI-arviointi osoittaa hoidon laadun ja tekee tehdyn työn näkyväksi, auttaa asettamaan tavoitteita ja osoittaa hoidon laadun ja vaikuttavuuden. Ohjelmiston avulla saadaan tärkeää tietoa hoitotyön kaikille tasoille hoitajasta johtajalle. (Raisoft 2022.)

2.3 Raisoft.net

Raisoftin päätuote on RAIsoft.net. Ohjelmistoa on kehitetty yli 20 vuoden ajan. Sitä myydään pääasiassa SAAS-ratkaisuna asiakkaille. Ohjelmisto on tehty modulaariseksi. Siinä on yli 50 moduulia, joista jokainen moduuli on sovellus suuremmassa RAIsoft.net-ohjelmistossa. Esimerkkejä moduuleista ovat lääkemoduuli, analysointi, yhteenvetoraportti, organisaation tilastot, käyttäjien hallinta jne.

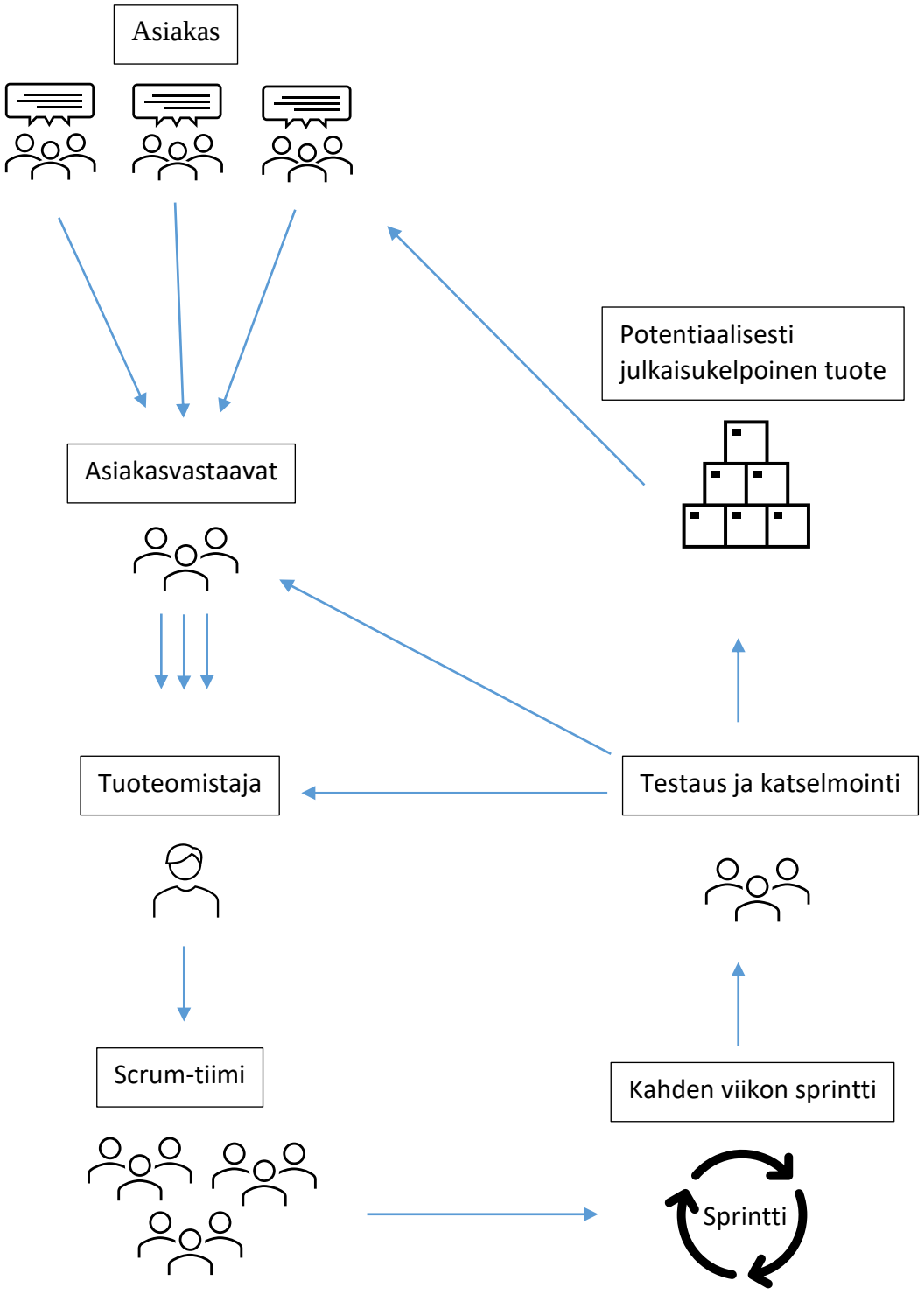
Kaikki asiakkaat käyttävät samaa tuotetta, mutta eri konfiguraatioilla. Moduuleita voi aktivoida tai deaktivoida asiakaskohtaisesti. Ohjelmistossa on myös laajennuksia (plugin). Näiden avulla ohjelmistossa voidaan toteuttaa asiakas- tai maakohtaisia lisävaatimuksia, kuten henkilötunnuksen tarkistuksen sääntöjä, asiakaskohtaisia raportteja yms. (Raisoft 2022.)

2.4 Asiakkaat ja kehitysprosessi

Vaatimukset ja toimeksiannot tulevat kehitystiimille asiakkailta eri asiakasvastaavien tai tuotteen omistajien kautta. Asiakkaat tilaavat uusia ominaisuuksia ohjelmistoon sekä ilmoittavat ohjelmassa esiintyvistä vioista Raisoftin asiakasvastaaville. Nämä asiakasvastaavat ilmoittavat tarpeista tuotteenomistajalle.

Raisoft käyttää Scrum-prosessia ohjelmistokehityksessä (KUVIO 1). Scrum on kevyt viitekehys, joka auttaa ihmisiä, tiimejä ja organisaatioita tuottamaan arvoa ratkaisemalla kompleksisia ongelmia joustavasti. Työ tehdään lyhyissä iteraatioissa, joita kutsutaan sprinteiksi. Sprintit ovat pituudelta aina vakioita. Yleisimmät sprintin kestot ovat kahdesta kuuteen viikkoon. Lyhyet sprintit mahdollistavat nopean oppimisen ja rajoittavat kustannuksiin ja työmäärään liittyvät riskit. (Schwaber & Sutherland 2020.)

Raisoftilla sprintin kesto on kaksi viikkoa. Tuote on potentiaalisesti julkaisukelpoinen jokaisen sprintin jälkeen. Tämä tarkoittaa, että sprintti ei saa sisältää tuotetta rikkovia muutoksia. Alla olevassa kuviossa nähdään Raisoftin kehitysprosessi. (Ketola 2022.)



KUVIO 1. SCRUM-prosessi Raisoftilla (mukaillen Ketola 2022).

2.4.1 Tuotteen omistaja

Tuotteen omistaja on henkilö, joka on vastuussa tuotteen arvon maksimoimisesta ja kehitystiimin työstä. Tuoteomistajan käytännön työtehtäviin kuuluu näin erityisesti tuotteen kehitysjonon hallinnointi ja eri sidosryhmien kanssa kommunikointi. Tuoteomistaja on Scrum-tiimin jäsen. (Schwaber & Sutherland 2020.)

Ketterässä ohjelmistokehityksessä tiimi on itse vetovastuussa tekemisestään sen sijaan, että heitä ohjataan ylhäältä päin. Tämän takia on tärkeä nähdä ero perinteisen projektipäällikön ja tuoteomistajan roolien välillä. (Schwaber & Sutherland 2020.)

Mitä tuotteenomistaja tekee

- On vastuussa projektin onnistumisesta
- Määrittelee suunnan tuotteelle
- Varmistaa, että projektilla on tarvittavat resurssit
- On aktiivisesti mukana tukemassa projektia
- Maksimoi tuotteen arvoa.
- Vastaa ohjelmiston kehitysjonosta.
- Ylläpitää kehitysjonon selkeässä muodossa ja näyttää Scrum-tiimille mitä kehitetään seuraavaksi.
- Vastaa kommunikoinnista muille sidosryhmille.
  Huolehtii siitä, että eri sidosryhmät ovat tietoisia tavoitteista, kuten esimerkiksi julkaisupäivistä tai projektin tämänhetkisestä tilanteesta.

(Schuurman 2020.)

Mitä tuotteen omistaja ei tee

- Ei ole virkailija
  Tuotteen omistajan ei kuulu kerätä ehdotuksia ja vaatimuksia eri sidosryhmiltä. Sen sijaan tuotteen omistajalla on oltava oma visio, ja kerätä sidosryhmiltä palaute sille visiolle.
- Ei ole käyttäjätarinoiden kirjoittaja
  Käyttäjätarinoiden kirjoittaminen tuotteen kehitysjonoon ei ole tuotteen omistajan tehtävä.
  Tuotteen omistaja voi toki osallistua käyttäjätarinoiden kirjoittamiseen, mutta se ei varsinaisesti kuulu tuotteen omistajalle

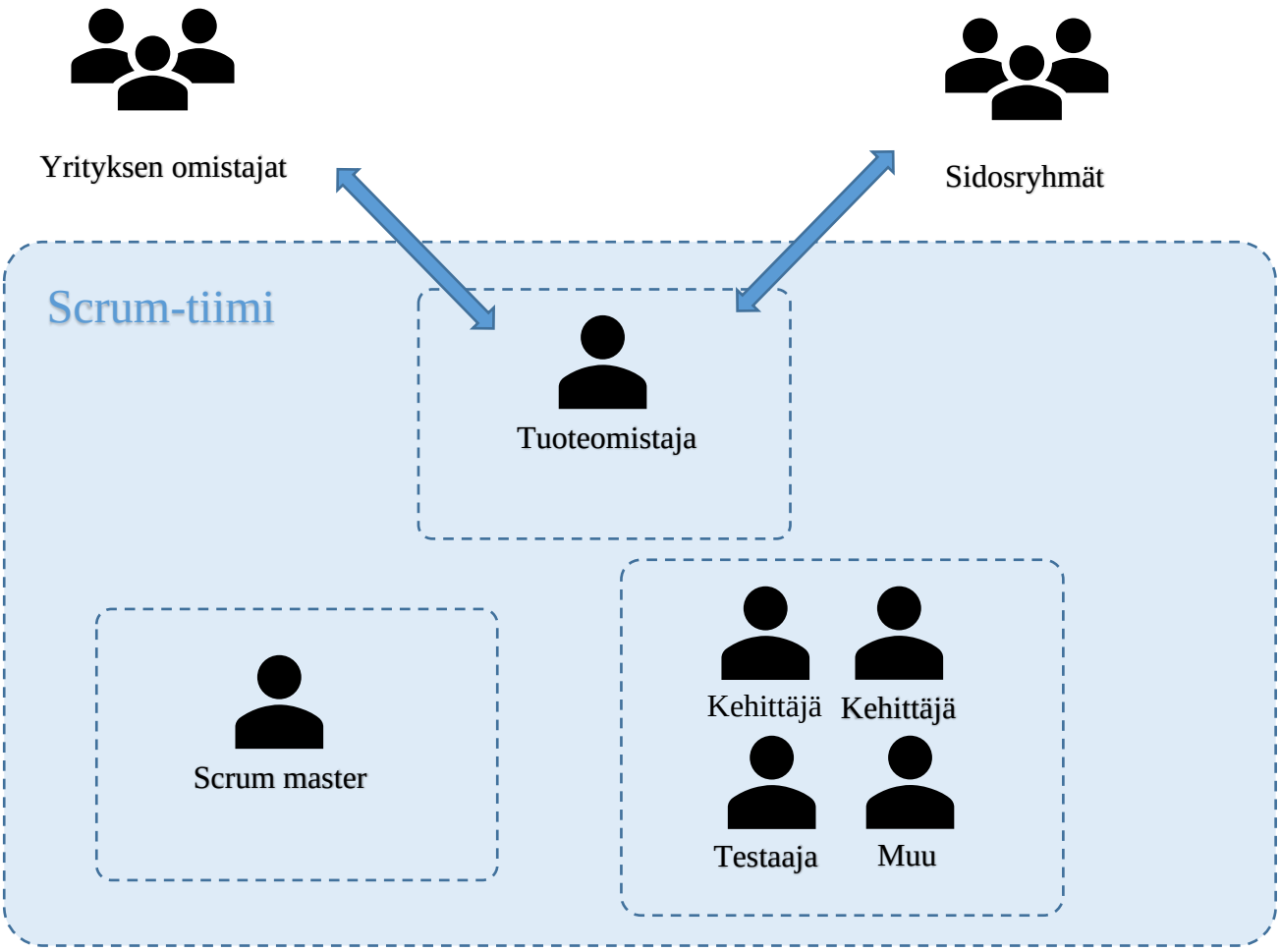
- Ei ole projektipäällikkö
  Projektipäällikkö ja tuotteen omistaja ovat kaksi eri asiaa. Tuotteen omistaja ei tee projektiraportteja, kaavioita tai muuta vastaavaa. Tuotteen omistaja ei myöskään kuulu mitata tiimin suorituskykyä eikä välittää kehitysprosessin optimoimisesta.
- Ei ole asiantuntija
  Tuotteen omistaja ei tarvitse olla kokenein henkilö tai asiantuntija.
- Ei ole portinvartija
  Scrum-tiimin jäsenet voivat kommunikoida sidosryhmille suoraan. Tuotteen omistaja ei ole portinvartija tiimin ja muiden sidosryhmien välissä.
- Ei ole esimies
  Tuotteen omistaja ei ole Scrum-tiimin esimies.

(Schuurman 2020.)

2.4.2 Scrum-tiimi

Scrum-tiimi on pieni tiimi, joka koostuu tyypillisesti korkeintaan kymmenestä henkilöstä (KUVIO 2). Scrum-tiimi on monialainen, tarkoittaen että tiimin jäsenillä on kaikki taidot, joita tarvitaan arvon tuottamiseen sprintissä. Tiimi on myös itseohjautuva, mikä tarkoittaa, että tiimi päättää itse kuka tekee, mitä ja milloin. Scrum-menetelmä kannustaa korkeatasoiseen kommunikaatioon tiimin jäsenten välillä, jotta tiimi voi tavoittaa yhteisiä tavoitteita sekä noudattaa samoja ohjeita ja sääntöjä. (Schwaber & Sutherland 2020.)

Scrum-tiimissä on myös rooli nimeltä Scrum master. Tämän henkilön tehtävänä on varmistaa, että tiimin päivittäinen työskentely on tuottavaa ja että pelisääntöjä noudatetaan. Scrum master vastaa tiimin tehokkuudesta. Hän palvelee tiimiä mm. valmentamalla tiimiä itseohjautuvuuteen, auttamalla keskittymään työtehtäviin ja poistamalla ulkopuolisen hälyn. (Schwaber & Sutherland 2020.)



KUVIO 2. SCRUM-tiimi (mukaillen Schwaber & Sutherland 2020).

2.4.3 Tuotteen kehitysjono

Tuotteen kehitysjono (Product backlog) on järjestetty lista kaikista tehtävistä, joita tuotteen parantamiseen tarvitaan. Se on Scrum-tiimin tärkein työlähde. Tuoteomistaja vastaa kehitysjonon sisällöstä ja priorisoinnista. Kehitysjono ei ole koskaan valmis, vaan se muuttuu muuttuvien vaatimuksien mukaan. Se sisältää kaikki ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset, jotka toteutetaan tuleviin tuoteversioihin. Kahden viikon aikana pyritään aloittamaan ja suorittamaan loppuun sprinttiin suunnitellut tehtävät. Sprintin aikana pyritään varmistamaan, että tiimin jäsenet voivat keskittyä työhön ja välttää ulkopuolisia häiriöitä. (Schwaber & Sutherland 2020.)

Tuoteomistaja vastaa kehitysjonosta, mutta koko Scrum-tiimi osallistuu kehitysjonon jalostamiseen. Jalostaminen tarkoittaa sitä, että lisätään tehtäville yksityiskohtia ja arvioidaan työmäärä. Varmistetaan myös, että tehtävät sisältävät kaiken sen tiedon, jota tarvitaan kehittämiseen. Kehitysjono saattaa sisältää useamman kuukauden työmäärän. Tehtäviä jalostetaan kuitenkin vain sen verran kuin tarvitaan seuraavaan sprinttiin. (Schwaber & Sutherland 2020.)

2.4.4 Sprintin kehitysjono

Sprintin kehitysjono (Sprint backlog) koostuu niistä tehtävistä, joita aiotaan tehdä seuraavan sprintin aikana. Tehtävien tulee olla valmiiksi jalostettuina niin pieniksi, että niitä voidaan toteuttaa valmiiksi sprintin aikana. Toisin kuin tuotteen kehitysjono, sprintin kehitysjono ei muutu sprintin aikana. Scrum-tiimi voi lisätä asioita sprintin kehitysjonoon, jos havaitaan, että uutta työtä tarvitaan kehitysjonon toteuttamiseksi. Tehtäviä voi myös poistaa Scrum-tiimin toimesta, jos niitä havaitaan tarpeettomiksi. Ainoastaan Scrum-tiimi voi siis muuttaa sprintin kehitysjonon sisältöä. (Schwaber & Sutherland 2020.)

2.4.5 Sprintti

Sprintit ovat kiinteitä yhden kuukauden mittaisia tai lyhyempiä tehtäväjaksoja. Jokainen sprintti on aina saman mittainen. Uusi sprintti alkaa välittömästi edellisen sprintin päädyttyä. Sprintit ovat lyhyitä, sillä tarkoituksena on varmistaa edistyminen ja parantaa ennustettavuutta. Monimutkaisuus ja riskit kasvavat, jos sprintti on liian pitkä. Sprintin aikana pyritään aloittamaan ja suorittamaan loppuun sprinttiin suunnitellut tehtävät. Sprintin aikana pyritään varmistamaan, että tiimin jäsenet voivat keskittyä työhön ja välttää ulkopuolisia häiriöitä. (Schwaber & Sutherland 2020.)

Raisoftilla sprintin kesto on kaksi viikkoa (Ketola 2022).

2.4.6 Testaus ja katselmointi

Sprintissä tehdyt työt testataan Scrum-tiimissä sprintin aikana. Tämän jälkeen asiakasvastaavat ja joskus jopa asiakkaat itse pääsevät katselmoimaan uutta ohjelmistoversiota. Tässä yhteydessä Scrum-tiimi ja

sidosryhmät keskustelevat, mitä sprintissä kehitettiin ja mitä voidaan tehdä seuraavaksi arvon optimoimiseksi. Tämä toimii pohjana seuraaville tuoteversioille. Tämä palaveri on epämuodollinen, ja sen tarkoituksena on saada palautetta ja edistää vuoropuhelua. Palaverin yhteydessä voidaan tarkistaa kehitysjonon tilanne ja arvioida todennäköistä valmistumisajankohtaa perustuen tähänastiseen edistymiseen. (Schwaber & Sutherland 2020.)

2.4.7 Potentiaalisesti julkaisukelpoinen tuote

Jokaisen sprintin tulisi alkaa aikomuksella saada pieni määrä parannuksia seuraavaan tuotejulkaisuun. Tuotteen tulee olla täysin testattu ja katselmoitu sprintin jälkeen. Tuotteessa ei saa olla julkaisua estäviä virheitä. Tuote on silloin potentiaalisesti julkaisukelpoinen. (Berteig 2020.)

Asiakkaat eivät kuitenkaan ehkä pysty käsittelemään uutta ohjelmistoversiota jokaisen sprintin jälkeen, esimerkiksi kahden viikon välein. Tästä syystä jokaista iteraatiota ei välttämättä toimiteta asiakkaalle. Scrum-tiimi huolehtii kuitenkin siitä, että tuotteen laatu on hyvä, mikäli tuoteomistaja päättää julkaista version asiakkaalle. (Berteig 2020.)

3 NYKYTILANNE OHJELMISTON SUUNNITTELUPROSESSISSA

Muutokset tulevat yleensä pieninä parannuksina olemassa olevalle tuotteelle. Suuria uusia muutoksia tuotteeseen tulee harvoin. Tämä työ käsittelee niitä tilanteita, jossa vaatimuksena on kokonaan uuden moduulin suunnittelu tai olemassa olevan moduulin suurempi refaktorointi.

Asiakkaalta tulevia muutospyyntöjä on kohdistettava oikeaan moduuliin. Jos saadaan asiakkaalta runsaasti uusia vaatimuksia, muutokset lajitellaan niihin moduuleihin, joista käyttäjä voi olettaa ominaisuuksien löytyvän. Esimerkiksi jos asiakas tarvitsee tilastollista tietoa, tämä muutos tehdään tilastotraporttiin. Tämän lajittelun tekee yleensä ohjelmistoarkkitehti asiakasvastaavan tai tuotteen omistajan kanssa, mutta myös käyttäjät tai muut sidosryhmät voivat osallistua tähän.

Joskus tulee tarve luoda kokonaan uusia moduuleja tai kokonaan uudistaa olemassa oleva moduuli. Tässä yhteydessä tehdään tarkemmat suunnitelmat ja tässä opinnäytetyössä käsitellään pääosin näitä tilanteita.

Ohjelmiston suunnittelu lähtee Raisoftilla monesti graafisesta suunnittelusta, jolloin esimerkiksi keksitään idea hyvältä näyttävästä raportista tai ohjelmistomoduulista. Tämä näyttää hyvältä siihen tilanteeseen, johon se on suunniteltu, ja sitä voi olla helppo markkinoida asiakkaille. Se ei välttämättä toimi hyvin oikealla datalla, eikä välttämättä kata asiakkaiden oikeita tarpeita. Vaikka suunnitelma saattaa näyttää hyvältä, se ei välttämättä ole helppokäyttöinen eikä välttämättä noudata samoja suunnittelukriteerejä, joita ohjelmiston aiemmin tehdyt osat noudattavat.

Monet RAIsoft.net-moduulit on tehty vuosia sitten. Jos uudelle moduulille halutaan uudistaa graafista ilmettä, pitää ottaa huomioon olemassa olevia moduuleja. Tässä tilanteessa täytyy arvioida, kannattaako uutta graafista ilmettä ottaa käyttöön, sillä se tarkoittaa myös vanhojen moduulien muuttamista.

Tässä työssä pyrkimyksenä on luoda malli, jolla prototyyppien avulla voidaan testata ja varmistaa, että ollaan ratkaisemassa asiakkaan oikeat ongelmat, ja tehdä se tavalla, jolla ohjelmiston visuaalinen yhtenäisyys ja helppokäyttöisyys säilyvät.

4 VALITUT TEORIAT

Tässä työssä käytetään menetelmiä, jotka jo ovat yrityksessä käytössä, ja osa menetelmistä on valittu mukaan sillä perusteella, että niiden on arvioitu toimivan yrityksessä hyvin. Tämä työ keskittyy suunnittelun eri vaiheisiin ja menetelmiin, joita voi hyödyntää näissä eri vaiheissa. Osa teorioista on rajattu työstä pois. Esimerkiksi graafinen suunnittelu on jätetty vähemmälle huomiolle, sillä yrityksessä on jo olemassa graafinen ohjeistus.

On olemassa myös paljon teorioita ja menetelmiä, joita voidaan hyödyntää ennen varsinaisten määritysten tekemistä, kuten järkevyysanalyysi, tai eri tapoja, joilla voidaan yrittää ymmärtää ongelmaa paremmin. Ne on rajattu tämän työn ulkopuolelle.

4.1 Käyttäjätarinat

Yksi Scrum-prosessin pääperiaatteista on tuottaa jatkuvasti asiakkaille ohjelmistoon lisäarvoa lyhyissä sprinteissä. Käyttäjätarina on loppukäyttäjän näkökulmasta kirjoitettu tarve tai toive ohjelmistoon. Näiden tarinoiden avulla voidaan ymmärtää projektin tavoite, mutta on myös tärkeää ymmärtää ongelmat eri käyttäjien näkökulmista. Monesti alustavat suunnitelmat eivät ota huomioon loppukäyttäjä. (Klimczak 2013.)

4.1.1 Käyttäjätarinan muoto

Käyttäjätarinoiden muoto on seuraava:

<*Käyttäjätyyppi*> haluan <*mitä*> koska <*miksi*>

Esimerkiksi:

Sairaanhoitajana haluan nähdä lukitsemattomat arvioinnit minun yksikössäni kirjautumisen jälkeen, koska haluan jatkaa arvioinnin tekemistä

<miksi> -on tärkeä, koska se antaa meille tiedon, jonka avulla voimme mahdollisesti ehdottaa parempaa ratkaisua ongelmaan. Tämän lisäksi se antaa tulevaisuutta varten tiedon miksi jokin ominaisuus lisättiin edelliseen versioon. Huonona esimerkkinä voi pitää esimerkiksi “Haluan korostaa tätä nappia punaisella värillä”, koska käyttäjätarinasta puuttuu perustelu. (Koivumäki-Lindholm 2013.)

Käyttäjätarinassa halutaan siis tietää seuraavat asiat:

- Käyttäjätyyppi
  Tämä määrittelee missä roolissa käyttäjä työskentelee
- Tarve/tavoite
  Mitä ominaisuuksia tai muutoksia käyttäjä tavoittelee
- Lisäarvo
  Määrittelee miksi tämä ominaisuus tuottaa käyttäjälle lisäarvoa

(Pokharel 2020.)

Käyttäjätarina tulisi kirjoittaa käyttäjän näkökulmasta, ei kehittäjän näkökulmasta. Käyttäjätarina kuten esimerkiksi ”Kaikkien autentikointien pitäisi tulla saman rajapinnan kautta” pitäisi välttää. Tämä käyttäjätarina on kehittäjän näkökulmasta ymmärrettävä, mutta ei käyttäjän näkökulmasta. (Pokharel 2020.)

4.1.2 Hyvien käyttäjätarinoiden piirteitä

Jokaisen käyttäjätarinan tulisi sisältää käyttäjän rooli, mitä käyttäjä haluaa ja miksi.
Hyvä käyttäjätarina noudattaa INVEST-mallia:

- Itsenäinen (Independent*)*
  Käyttäjätarinat eivät saa olla toisistaan riippuvaisia. Jos käyttäjätarinat ovat riippuvaisia toisistaan, voi syntyä ongelmia suunnittelussa.
- Neuvoteltavissa (Negotiable)
  Käyttäjätarinat eivät ole sopimuksia. Käyttäjätarina voi joustaa, kun ymmärrys ongelmasta kasvaa. Tavoitten tulee olla selkeä, mutta samalla mahdollistaa keskustelua, jos löydetään parempia ratkaisuja.
- Arvokas (Valuable)
  Käyttäjätarinan tulisi esittää, miten muutos tuottaa lisäarvoa käyttäjälle.
- Arvioitavat (Estimable)
  Käyttäjätarinan tulisi olla tarpeeksi selkeä, jotta sen työmäärää voidaan arvioida. Käyttäjätarinat, jotka ovat epäselvät, tai liian laajoja ei välttämättä saa korkeaa prioriteettia.

- Pieni (Small)
  Käyttäjätarinoiden tulisi olla pieniä, eivätkä ne saa vaatia liikaa kehitysaikaa. Pienet käyttäjätarinat auttavat tiimiä fokusoimaan, tuottamaan käyttäjälle lisäarvoa ja mahdollistavat nopean palautteen.
- Testattava (Testable)
  Käyttäjätarinan on oltava helposti ymmärrettävä, jotta testaus on mahdollista suorittaa.

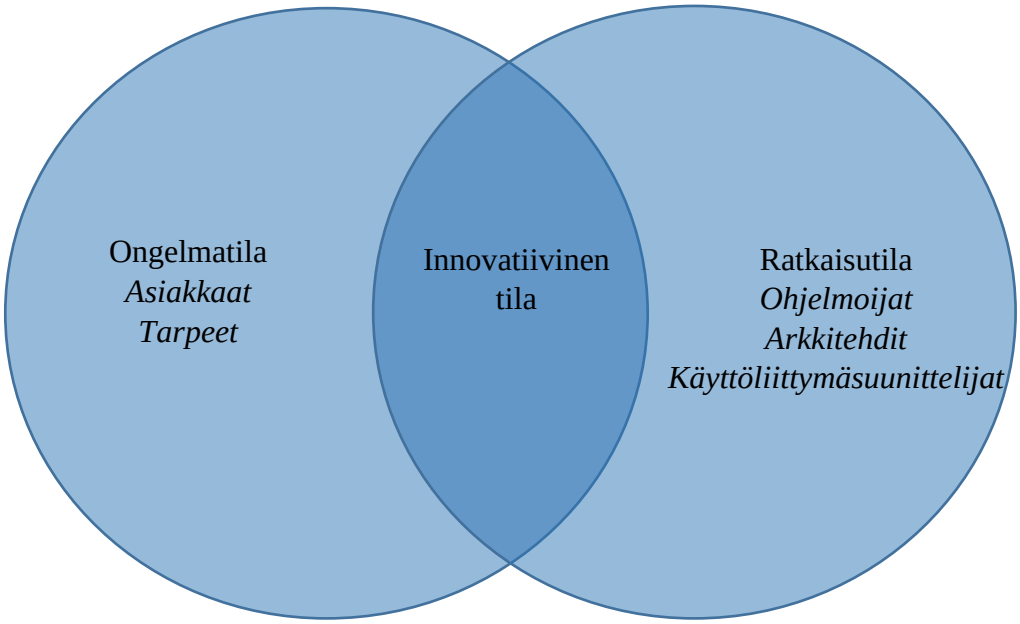
4.1.3 Haasteet

Käyttäjätarinoita käytetään ohjelmiston vaatimuksien ja lisäarvon määrittelemiseen. Käyttäjätarina ei pelkästään kuvaa sitä, mitä pitää kehittää, vaan se määrittelee, mikä on tärkeää käyttäjän näkökulmasta. Koska käyttäjätarina on kirjoitettu loppukäyttäjän näkökulmasta, sen pitäisi luoda lisäarvoa asiakkaalle. Tästä syystä käyttäjätarinoita tulee kirjoittaa siten, että lisäarvo on selkeästi ymmärretty.

Laadukkaan käyttäjätarinan kirjoittaminen on haastava. Pokharelin (2020) mukaan vain harva ketterää kehitysmenetelmää noudattava tiimi hyödyntää käyttäjätarinoita. Tutkimuksessa vain 45 % osallistujista käytti käyttäjätarinoita. Tämän lisäksi tästä joukosta vain 14,81 %:lla oli hyvä ymmärrys siitä, mikä on käyttäjätarina.

4.1.4 Innovatiivinen tila

Käyttäjätarinat eivät ole teknisiä. Niitä on helppo lukea ja helppo ymmärtää. Käyttäjätarinalla pyritään ymmärtämään ongelma käyttäjän tai asiakkaan näkökulmasta. Ne toimivat rajana ongelman ja ratkaisun välillä (KUVIO 3). Jos asiakkaalta saadaan esimerkiksi valmis tekninen kuvaus, se estää ohjelmistosuunnittelijoita hyödyntämästä heidän ammattitaitoansa ja jättää vähemmän tilaa innovaatiolle. Käyttäjätarina jättää siis tilaa innovaatiolle ja mahdollistaa eri ammattiryhmien käyttää heidän ammattitaitoansa. (O'hEocha & Conboy. 2010.)



KUVIO 3. Innovatiivinen tila (mukaillen O'hEocha & Conboy 2010).

4.1.5 Epic

Joskus tarvitaan isompia ominaisuuksia, joita ei voida kuvata pelkästään käyttäjätarinalla. Epic on laajempi kokonaisuus, kuten ohjelmistostrategia tai projekti, joka sisältää joukon yksittäisiä käyttäjätarinoita (KUVIO 4). Epicin avulla voidaan helpommin hallita laajempia kokonaisuuksia. (ProductPlan.)

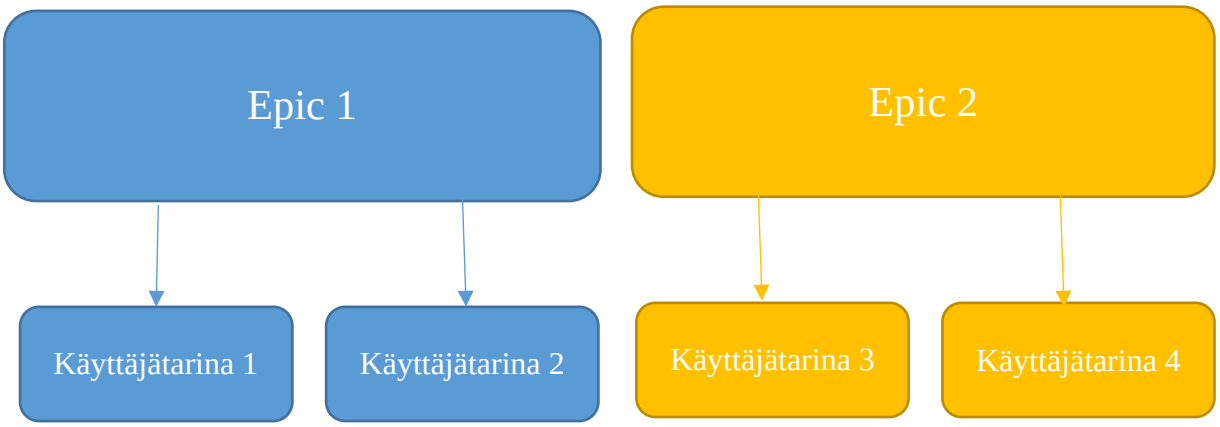
Epicin olisi hyvä sisältää ainakin seuraavat tiedot:

- Nimi
  Kuvaava nimi, kuten esimerkiksi ”Tiedonsiirron suorituskyvyn parantaminen”
- Kuvaus
  Kuvaus siitä, mitä halutaan saavuttaa epicillä. Tämä voi noudattaa samaa kaavaa kuin käyttäjätarina, <*Käyttäjätyyppi*> haluan <*mitä*> koska <*miksi*>. Tämän lisäksi on myös hyvä parilla lauseella kirjoittaa taustasyyt epicille.
- Laajuus
  Määritellään tarkasti mitä tämä epic koskee. Jos esimerkiksi tavoitteena on suorituskyvyn parantaminen, kirjoitetaan mihin kohtiin tämä liittyy.

- Hyväksymiskriteerit
  Määritellään mitkä ovat ominaisuuden hyväksymiskriteerit.
- Puretaan epic käyttäjätarinoiksi
  Kun on ymmärrys, mitä halutaan saavuttaa epicillä, voidaan purkaa se käyttäjätarinoihin.

(ProductPlan.)

Epicejä ei tyypillisesti voida toteuttaa yhden sprintin aikana, vaan ne vaativat useamman sprintin. Jokaiselle sprintille otetaan sen verran käyttäjätarinoita kuin voidaan toteuttaa sprintin aikana. Epic on valmis, kun kaikki siihen liittyvät käyttäjätarinat on tehty. (ProductPlan.)



KUVIO 4. Epic ja käyttäjätarinat (mukaillen Rehkopf 2022).

4.1.6 Hyväksymiskriteerit

Täydellisessä maailmassa käyttäjätarinat riittäisivät ongelman ymmärtämiseen ja luodun ratkaisun testaamiseen. Joskus voi kuitenkin olla tarve kirjoittaa myös hyväksymiskriteerit, joiden pitää täyttyä, jotta käyttäjätarinaan perustuva ongelma voidaan katsoa ratkaistuksi. Hyväksymiskriteerit eivät ole tärkeitä pelkästään testaamisen apuvälineinä, vaan niiden avulla saadaan myös tietää, onko tarve ymmärretty oikein. (Pichler 2016, Maryna Z 2020).

Käyttäjätarina:

Käyttäjänä haluan pystyä rekisteröitymään verkkokauppaan, jotta voisin tehdä ostoksia

Hyväksymiskriteerit:

- Kaikkien kenttien tulee olla syötettynä
- Käyttäjän sähköpostin muodon tulee olla kelvollinen
- Samasta IP-osoitteesta voi rekisteröityä enintään 3 kertaa tunnissa

Valmiin määritelmä (definition of done tai DoD) on laajempi käsite, joka kuvaa sitä, milloin voidaan pitää työtä tehtynä. Siinä missä hyväksymiskriteeri yleensä käsittelee yksittäistä ominaisuutta, valmiin määritelmä käsittelee yleensä laajemmin ohjelmistoa tai epiciä. (Tech Agilist.)

Hyödyt

- Selkeä määritelmä lisää läpinäkyvyyttä ja johtaa viime kädessä parempaan laatuun.
- Sprintin suunnittelun aikana tiimi voi kehittää tai vahvistaa määritelmän, jonka avulla on helpompi arvioida jäljellä olevan työn määrää

(Tech Agilist.)

4.2 Pienin toimiva tuote – MVP

Pienin toimiva tuote (Minimum Viable Product, MVP) on versio tuotteesta, jossa on vain sen verran ominaisuuksia, jotta siitä saadaan käyttökelpoista palautetta asiakkailta, ei muuta. Yleinen harhaluulo on se, että MVP:tä pidetään lopullisena tuotteena. MVP-versiolla voidaan saada asiakkailta palautetta aikaisessa vaiheessa ennen kuin lisätään ominaisuuksia. Tämä puolestaan vähentää riskejä ja hukkaa. (Kartusek 2020.)

Yhdistämällä MVP-kehittäminen ketterän kehityksen menetelmiin voidaan jatkuvasti tuottaa uusia versioita, joissa jokainen iteraatio on toimiva tuote, ja jokainen vie tuotetta lähemmäksi lopullista päämäärää (Kartusek 2020). Tätä havainnollistetaan kuviossa 5.

KUVIO 5. Pienimmästä toimivasta tuotteesta monipuolisempaan tuotteeseen (Kartusek 2020).

4.2.1 Yleisiä haasteita

Monet saattavat olettaa, että MVP on lopullinen tuote tai beta-versio. MVP on kuitenkin vain tuotteen seuraava vaihe, jolla saadaan asiakkailta oikeanlaista palautetta.

Seuraavia virheitä saatetaan tehdä, kun suunnitellaan pienin toimiva tuote eli MVP

- **Oletus, että MVP on lopullinen tuote**
  Yleinen harhaluulo on se, että MVP:tä pidetään lopullisena tuotteena. Tämän seurauksena pyritään laittamaan kaikki ominaisuudet, joita oletetaan, että lopullisessa tuotteessa tulisi olla. Käytetään esimerkiksi turhan paljon aikaa käyttöliittymän viilaukseen.
- **Minimalistinen suunnittelu**
  Kun yritetään karsia ominaisuuksia, voidaan vahingossa karsia juuri niitä ominaisuuksia, joita asiakkaat tarvitsevat. Kun suunnitellaan pienintä mahdollista tuotetta, pitää myös varmistaa, että suunnitellaan toimivaa tuotetta.
- **Puutteellinen markkinatutkimus**
  Ei ole välttämättä mietitty sitä, onko tuotteelle oikeasti tarve.

(Brych 2018.)

4.3 Prototyypit

Prototyyppien tarkoitus on saada käyttäjiä kokeilemaan tuotetta sen sijaan, että käyttäjä yrittäisi arvioida lopputulosta pelkästään käyttäjätarinoiden ja kuvauksien perusteella. Prototyyppien avulla voidaan vähentää aikaa ja kustannuksia, koska niiden avulla voidaan tunnistaa ongelmia mahdollisimman aikai-

sessa vaiheessa. Prototyyppien avulla voidaan myös osallistaa eri sidosryhmiä ja siten parantaa kommunikointia. Koska asiakas tuntee omat tarpeensa parhaiten, parannettu kommunikointi voi johtaa parempaan laatuun ja asiakastyytyväisyyteen. (Klimczak 2013.)

4.3.1 Karkeat prototyypit

Projektin aikaisessa vaiheessa ei kannata käyttää liikaa aikaa tarkkoihin suunnitelmiin tai tarkkoihin prototyyppeihin. Jos käyttää paljon aikaa ja vaivaa suunnitelmiin, suunnitelmista on vaikeampi luopua, mikäli huomataan että ratkaisu ei ole tyydyttävä. Karkean prototyypin avulla voidaan tuoda idea esiin helposti ja vaivattomasti. (Miller 2011.)

Tätä varten voi tehdä karkeita ns. ”low fidelity” -prototyyppejä. Prototyypin voi piirtää esimerkiksi paperille, piirtotaululle tai käyttää apuna PowerPoint- tai muuta vastaavaa ohjelmistoa. Näiden nopeiden prototyyppien avulla voidaan toteuttaa useita iteraatioita, kunnes saavutetaan riittävän hyvä ratkaisu. Hyvä tapa voi myös olla tehdä A- ja B-vaihtoehtoja. Näistä karkeista prototyypeistä on myös helppo luopua, mikäli todetaan että ratkaisu ei ole tyydyttävä. (Miller 2011.)

Karkeiden prototyyppien hyödyt:

- Nopea tehdä verrattuna tarkempaan prototyyppiin
- Helppo muuttaa, mikäli prototyypissä huomataan puutteita
- Ei vaadi teknistä osaamista, kuka tahansa voi osallistua
- Fokusointi kokonaiskuvaan
    - Graafiset suunnittelijat eivät tuhlaa aikaansa mm. lopulliseen värimaailmaan
    - Asiakkaat ja sidosryhmät antavat luovia ideoita, sillä heillä ei ole mahdollisuutta saivarteluelle.

(Miller 2011.)

Mikäli prototyyppiä esitellään tiimin ulkopuolisille henkilöille, tulisi varmistaa, että osallistuvaa henkilöä ei johdatella haastattelijan haluamalle lopputulokseen. Prototyypin testauksen tavoitteena on saada käyttökelpoista palautetta testin osallistujalta. (Miller 2011.)

Karkeiden prototyyppien avulla saadaan tietoa seuraavista asioista:

- Konseptuaalinen malli. Ymmärtääkö käyttäjä mistä on kyse?

- Toiminnallisuus. Kattaako se tarpeita? Sisältääkö tuote kaikki tarvittavat ominaisuudet?
- Navigointi. Osaavatko käyttäjät käyttää tuotetta?
- Termistö. Ymmärtävätkö käyttäjät termejä ja tekstejä?
- Näytön sisältö. Mitä näytöllä tarvitsee olla?

(Miller 2011.)

Karkeista prototyypeistä ei kuitenkaan saada kaikkia tietoja. Esimerkiksi lopullisesta värimaailmasta ei saada tietoja. Myöskään käytön tehokkuudesta tai ohjelmiston nopeudesta, kuten esimerkiksi latausnopeudesta ei saada tietoa. Myös pienet yksityiskohdat voi puuttua, ellei niitä ole otettu mukaan prototyyppiin. (Miller 2011.)

Karkeiden prototyyppien käyttäminen on hyödyllistä projektin aikaisessa vaiheessa, jossa halutaan nopeasti kokeilla eri ideoita. Koska prototyyppi on hyvin karkea, se estää tiimin ja sidosryhmien huomion kiinnittymisen epäolennaisiin asioihin, kuten fontteihin, asetteluun jne. Tästä syystä prototyypin kuulukin olla karkea. (Miller 2011; Klimczak 2013.)

4.3.2 Tarkat prototyypit

Tarkempi High Fidelity -prototyyppi on tietokoneella tehty prototyyppi. Prototyypin voi tehdä esimerkiksi Adobe XD -sovelluksella. Prototyypin olisi hyvä olla klikattava ja kokeiltava tarkemman käyttäjäpalautteen saamiseksi. Prototyypin tekeminen vie aikaa, minkä lisäksi prototyypin testauksen jälkeen joudutaan yleensä tekemään muutoksia prototyyppiin. Tästä syytä prototyypistä ei välttämättä kannata tehdä valmiin tuotteen näköistä. Prototyyppi voi olla esimerkiksi mustavalkoinen. Prototyypin ei tarvitse sisältää tässä vaiheessa kaikkia graafisia elementtejä. (Miller 2011; Klimczak 2013.)

Mikäli vaadittu ominaisuus on hyvin graafinen, esimerkiksi kaavio tai visuaalinen ilme, prototyypin pitää olla hyvin lähellä lopullista tuotetta, jotta saadaan oikeanlaista palautetta. Prototyyppien avulla voidaan selvittää, tehdäänkö oikeat asiat ja osaavatko käyttäjät käyttää tuotetta. Prototyypin avulla voidaan myös testata, toimiiko visio ylipäätään eli onko tuotteelle tai ohjelmistomuutokselle tarvetta. (Miller 2011; Klimczak 2013.)

Tarkan prototyypin avulla voidaan saada selville seuraavia asioita:

- Samat asiat, jotka voidaan saada selville karkeiden prototyyppien avulla

- Graafinen ulkoasu
- Värit, fontit, ikonit
- Interaktiiviset palautteet kuten painikkeiden klikkaus ja palaute
- Käytön tehokkuus. Ovatko painikkeet ja tekstit esimerkiksi tarpeeksi isot?

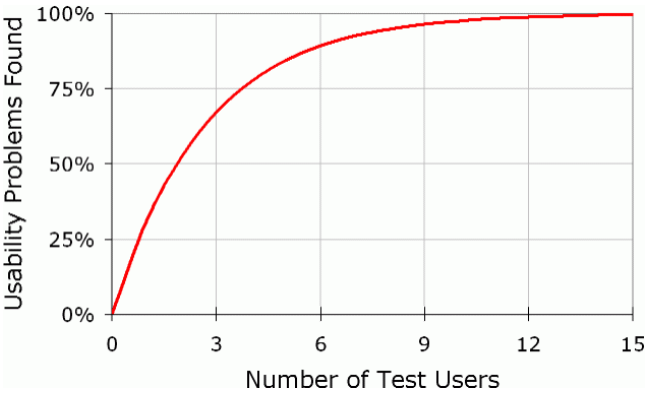
(Miller 2011; Klimczak 2013.)

Nopeus, esimerkiksi latausnopeus, ei näy tarkemmassakaan prototyypissä. Tarkempi prototyyppi avaa mahdollisuuden prototyypin testaajille keskittyä epäolennaisiin asioihin kuten värimaailmaan. Tarkemman prototyypin kehittämiseen käytetty työkalu ei myöskään saa ohjata graafista suunnittelua. Tarkemman prototyypin tekeminen vaatii enemmän työtä kuin karkea prototyyppi. Tarkkojen prototyyppien avulla voidaan kuitenkin oppia asioita, joita ei voida oppia karkeilla prototyypeillä. Kun esimerkiksi prototyyppi on klikattava, sen avulla voidaan helpommin suorittaa käyttäjätestejä. (Miller 2011; Klimczak 2013.)

4.4 Prototyypin käyttäjätestaus

Käyttäjätestauksen tarkoituksena on löytää tuotteesta käytettävyysongelmia. Testaukseen käytetään kohdehenkilöitä, jotka edustavat kohdeyleisöä. Henkilölle kerrotaan mistä on testauksessa kyse ja korostetaan, että ei haluta testata kohdehenkilön kykyä ymmärtää prototyyppiä vaan nimenomaan etsitään puutteita prototyypistä. Testejä ja heuristisia arvioita voidaan tehdä projektin eri vaiheessa. Projektin eri vaiheissa voidaan puntaroida, mikä testi sopii parhaiten. (Luoma 2018.)

Jo 3-5 käyttäjää riittää laadulliseen testaukseen. Mikäli halutaan tilastollista tietoa, tarvitaan kymmeniä henkilöitä (KUVIO 6). Esimerkiksi viidellä henkilöllä voidaan löytää yli 75 % laadullisista ongelmista. Ennen laajempaa testausta voidaan suorittaa pilottitestaus, jolla löydetään virheitä itse testisuunnitelmasta. (Nielsen 2000.)



KUVIO 6. Testikäyttäjien määrä verrattuna löydettyihin käytettävyysongelmiin (Nielsen 2000).

4.4.1 A/B-testaus

Nimensä mukaan A/B-testaus on vertailua, jossa etsitään paras vaihtoehto. Voidaan tehdä kaksi variaatiota samasta ehdotuksesta ja selvittää kumpaa niistä pidetään parempana. Vertailusta voidaan johtaa tilastollinen ero. (Young 2014.)

A/B-testauksia käytettiin ainakin jo 1960-luvulla, kun Bell Systems testasi useita eri variaatioita puhelimestaan. Näissä testeissä kysyttiin käyttäjiltä, millaiset painikkeet ovat mukavimmat käyttää puhelimessa. Tätä lähestymistapaa käytetään laajalti, koska tämä menetelmä tarjoaa suoraa näyttöä käyttäjien käyttäytymisestä. Ohjelmistoa voidaan pitää ns. ”ikuisena beta-versiona” (perpetual beta), jossa tuotetta parannetaan jatkuvasti eri iteraatioissa, joiden pyrkimyksenä on löytää paras mahdollinen lopputulos. (Young 2014.)

4.4.2 Testitehtävät ja käsikirjoitus

Ennen testausta tehdään testitehtävät ja käsikirjoitus. Tehtävien tulisi vastata tuotteen oikeita käyttötapauksia eli niitä ominaisuuksia, joita ollaan kehittämässä. Tehtävien tulisi olla kuvailevia, ei johdattelevia. Esimerkiksi: ”Jätit arvioinnin kesken. Kirjaudu ohjelmaan ja tee arviointi loppuun”. (Nielsen 2000.)

Käyttäjähaastattelun käsikirjoitus toimii perustana, kun keskustellaan käyttäjän kanssa. Käsikirjoituksessa esitetyt kysymykset ohjaavat testaajaa keskustelun aikana. Sen avulla voidaan varmistaa, että testi on sama riippumatta siitä, kuka suorittaa testin. Esimerkki käsikirjoituksesta liitteessä 1. (Lam 2020.)

4.4.3 Testin toteutus

Testi olisi hyvä nauhoittaa. Tällä tavalla testin vetäjä voi keskittyä testin vetämiseen sen sijaan, että keskittyy merkintöjen kirjaamiseen. Testitilanteessa pyydetään testiosallistujaa myös ajattelemaan ääneen. Osallistuja voi siis kertoa esimerkiksi miltä hänestä tuntuu ja mitä hän ajattelee, kun yrittää suorittaa tehtävän. Testin vetäjän tulisi pysyä hiljaa ja antaa testiosallistujan yrittää ratkaista ongelman itse. Testin alussa voi kysyä pari lämmittelykysymystä. Varsinaisen testikysymyksen jälkeen voi myös kysyä täydentäviä kysymyksiä, esimerkiksi: ”Kuinka vaikeaa se oli”. (Lam 2020. McCracken 2016.)

4.4.4 Käyttäjätestauksen raportointi

Myös omasta raportista pitää tehdä selkeä. Tulokset voi ryhmittää esimerkiksi seuraavasti:

- (T) Tekninen ongelma
- (0) Kosmeettinen ongelma
- (1) Käytettävyysongelma: haittaa käyttöä, mutta ei mahdoton käyttää
- (2) Kriittinen ongelma: jollei ongelmaa korjata, käyttäjät eivät voi käyttää tuotetta.

Raportin tulisi sisältää testitehtävät, havaitut ongelmat sekä ongelman jaottelun. Korjausehdotukset on hyvä pitää erillään virheistä tässä vaiheessa, vaikka korjaus vaikuttaisikin selkeältä. Raportissa kannattaa pitää fokus ongelmassa ja huomioida myös positiiviset palautteet eli toimivat asiat, joita ei haluta rikkoa korjatussa versiossa. (Isherwood 2018. Usability.)

4.4.5 Tarkemmat määritykset

Kun käytettävyystestaus on suoritettu, voidaan kirjoittaa tarkemmat määritykset. Mikäli kyseessä on esimerkiksi raportointimoduuli, määritetään tarkat logiikat ja säännöt. Tällöin esimerkiksi määritellään mitkä valinnat on mahdollista valita pudotusvalikosta ja niin edelleen. (Isherwood 2018. Usability.)

4.4.6 Testauksen jälkeen

Testauksen jälkeen tehdään uusi prototyyppi, jossa on korjattu mahdollisimman monta käytettävyysongelmaa. Tämän jälkeen testataan uudestaan. Mikäli testattava tuote on jo julkaistu ja käytössä, arvioidaan mitkä käytettävyysongelmat ovat tarpeeksi isoja, jotta niitä kannattaa korjata. (Isherwood 2018. Usability.)

4.5 Graafinen suunnittelu

Graafisen suunnittelun tavoite on ilmaista käyttäjälle selkeästi ja yksinkertaisesti mitä on tarjolla ja se saa olla myös kivan näköistä. Ihmisen mieli pystyy käsittelemään vain rajallisen määrän tietoa kerrallaan. Selkeä asettelu tuo rauhaa ja ammattimaisuutta. Ihmisillä on myös taipumus muodostaa asioista kokonaisuuksia. Esimerkiksi vierekkäin aseteltuina asiat mielletään kokonaisuudeksi. (Chapman 2018.)

Yhtenä menetelmänä graafisessa suunnittelussa on poistaa kaikki se, mikä ei ole tarpeen. Tällä menetelmällä on kolme vaihetta:

1. Määrittele minkä ominaisuuksien pitäisi sisältyä ohjelmistoon.
2. Arvioi kriittisesti kaikki elementit ja määrittele ovatko ne oikeasti tarpeellisia.
3. Poista ominaisuudet, mikäli ne eivät ole tarpeellisia. Tiettyjä ominaisuuksia voi kokeilla poistaa, vaikka niitä pidetäänkin tarpeellisina.

(Chapman 2018.)

Tasapainoisen värityksen löytämiseksi löytyy erilaisia menetelmiä. Esimerkiksi sivua voi katsoa siristämällä silmiä (Squint test). Tällä tavalla voi huomata elementtejä, jotka esimerkiksi ovat ylikorostettuja. (Chapman 2018.)

4.5.1 Fittin laki (Fitt’s law)

Fittin laki käsittelee aikaa, joka kuluu siihen, kun käyttäjä siirtää osoitinta (hiiri) johonkin toiseen näytössä olevaan elementtiin. Käyttäjän interaktion nopeus perustuu siihen, kuinka kaukana elementit ovat toisistaan sekä niiden kokoon. Jos painike on kaukana ja pieni, kestää pitempään käyttäjältä päästä sinne

ja osua siihen. Suuret painikkeet helpottavat tätä. Myöskin elementit, jotka ovat lähellä hiiren osoitinta, esimerkiksi alasvetovalikko tai oikeaklikkaus, nopeuttavat käyttöä.

Näyttöalueen reunat toimivat rajoittimena, jonka yli hiiren osoitin ei voi liikkua. Esimerkkinä voi pitää Windows-ohjelmistojen X-painiketta. Hiiren osoitin osuu siihen painikkeeseen aina, jos hiirtä vie reilusti yläoikeaan nurkkaan. Tämä siis tekee painikkeesta teoriassa äärettömän ison. Tätä voi hyödyntää selaimessa esimerkiksi laittamalla painikkeita kiinni vasempaan laitaan. (Fitt 1954.)

4.5.2 Hick-Hymanin laki

Hick-Hymanin laki kertoo siitä, miten informaation entropia vaikuttaa käyttäjän nopeuteen tehdä valintoja. Kun käyttäjälle tarjotaan useita vaihtoehtoja, on ohjelmisto hitaampi käyttää. (Hick 1952.)

Tiettyjä poikkeuksia on olemassa. Esimerkiksi pitkästä nimilistasta voi olla helppo löytää haluamansa henkilö, mikäli lista on järjestetty aakkosittain. Tässä käyttäjän katse kohdistuu ensin sattumavaraiseen kohtaan listassa ja käyttäjän on helppo arvioida, onko hänen haluamansa henkilön nimi ylempänä vai alempana. (Hick 1952.)

4.5.3 Klikkauksien määrä ei ole tärkeä

Käyttöliittymän suunnittelussa klikkauksien määrää on perinteisesti pidetty yhtenä tekijänä helppokäyttöisyydelle. Esimerkiksi vuonna 2001 julkaistiin *kolmen klikkauksen sääntö* (Laubheimer 2019). Tämän mukaan käyttäjä turhautuu, jos haluttu sivu tai tieto on enemmän kuin kolmen klikkauksen päässä.

Halutun sisällön saaminen mahdollisimman vähällä klikkauksilla ei ole pelkästään harhaanjohtavaa, vaan myös monesti käytännössä mahdoton toteuttaa. Käytettävyyden mittaamiselle ei tulisi perustua klikkauksien määrään, vaan käytettävyyttä tulisi testata eri menetelmillä. Tutkimuksien mukaan käyttäjät eivät turhaudu klikkauksien määrään, vaan siihen, etteivät he löydä haluamaansa. (Nielsen 2008. Laubheimer 2019. Rajeesh 2018.)

Klikkauksien määrän vähentäminen voi johtaa esimerkiksi siihen, että jokaiselle sivulle käyttöliittymässä lisätään laajat navigointivalikot. Tämä vähentää klikkauksen määrää, mutta samalla saattaa tehdä ohjelmistosta entistä sekavamman käyttää. (Nielsen 2008. Laubheimer 2019. Rajeesh 2018.)

4.6 Heuristinen arviointi

Sana heuristiikka juontaa juurensa Kreikan kielen ilmaisusta *heureka*, ”löysin”. Heuristinen arviointi käsittelee käytettävyyden yleisiä sääntöjä. Heuristinen arviointi on käytettävyysasiantuntijan arvio käytettävyydestä ja sillä voidaan korjata osa käytettävyysongelmista esimerkiksi ennen käyttäjätestausta. Käytettävyys-ammattilaiset ovat parempia löytämään ongelmia kuin ei-ammattilaiset. Lisäksi useampi ongelma voidaan löytää, jos useampi käytettävyysasiantuntija suorittaa arvioinnin. (Nielsen 1992.)

4.6.1 Nielsenin heuristiikka

On olemassa useita eri heuristisia ohjeita. Hyvin tunnettu ja ehkä eniten käytetty on Nielsenin heuristiikka. (Nielsen 1994.)

Nielsen heuristiikan top-10:

1. **Tuotteen tilan näkyvyys**
   Käyttäjän pitäisi aina pystyä nopeasti huomaamaan, mikä on tuotteen tila tai toiminto.
2. **Tuotteen ja tosielämän vastaavuus**
   Tuotteen ja sen ohjeistuksen tulisi käyttää tavallisia ja tuttuja termejä, sanontoja ja käsitteitä mieluummin kuin omaa erikoistermistöä.
3. **Käyttäjän kontrolli ja vapaus**
   Käytön pitäisi olla tuotteen käyttäjän määrättävissä.
4. **Yhteneväisyys ja standardit**
   Miten muut ohjelmat, joita käyttäjät myös käyttävät työssään, toimivat ja mitä termejä siellä on käytetty? Toimiiko sivusto niin kuin käyttäjä olettaa sen toimivan?
   Esimerkki: verkkokaupassa oletetaan, että ostoskori-painike löytyy yläpalkista oikealla.
5. **Virheiden estäminen**
   Edellyttääkö käyttö ohjeiden lukua tai koulutusta? Onko olemassa riski, että käyttäjä klikkaa väärää linkkiä, koska ymmärsi väärin?

6. **Tunnistaminen mieluummin kuin muistaminen.**
   Onko eri sivut toteutettu samalla tavalla? Onko esimerkiksi etsi ja tallenna -painikkeet toteutettu samalla tavalla ohjelman eri osissa?
7. **Käytön joustavuus ja tehokkuus**
   Käytön tulisi olla joustavaa ja tehokasta sekä aloitteleville että edistyneille käyttäjille. Ovatko yleisimmät toiminnot helposti käytettävissä?
8. **Esteettinen ja minimalistinen suunnittelu**
   Tuotteessa tulisi olla vain halutun tiedon, toiminnot, tunnelman ja tyylin ilmaisevat muodot, ei enempää.
9. **Virhetilanteiden tunnistaminen, ilmoittaminen ja korjaaminen**
   Virheilmoitusten tulisi selvitä helposti: mitä tapahtui, miksi näin kävi, miten asia voidaan korjata ja kuinka se voidaan välttää ensi kerralla. Onko virheilmoitus ymmärrettävissä?
10. **Opastus ja ohjeistus**
    Onko ohjeistus helposti saatavilla?

(Nielsen 1994.)

4.6.2 Vaiheet

Heuristinen arviointi on helppo toteuttaa. Työn suorittaa käyttöliittymäasiantuntija, ja hän vertaa käyttöliittymää heuristisiin sääntöihin. Lopuksi hän listaa ja perustelee ongelmat. (Miller 2011.)

Vaiheet:

- Käyttöliittymää tarkistetaan perusteellisesti
- Verrataan käyttöliittymää heuristisiin sääntöihin
- Listataan käytettävyysongelmat
- Esitetään ja perustellaan jokainen ongelma hyödyntäen heuristisia sääntöjä

(Miller 2011.)

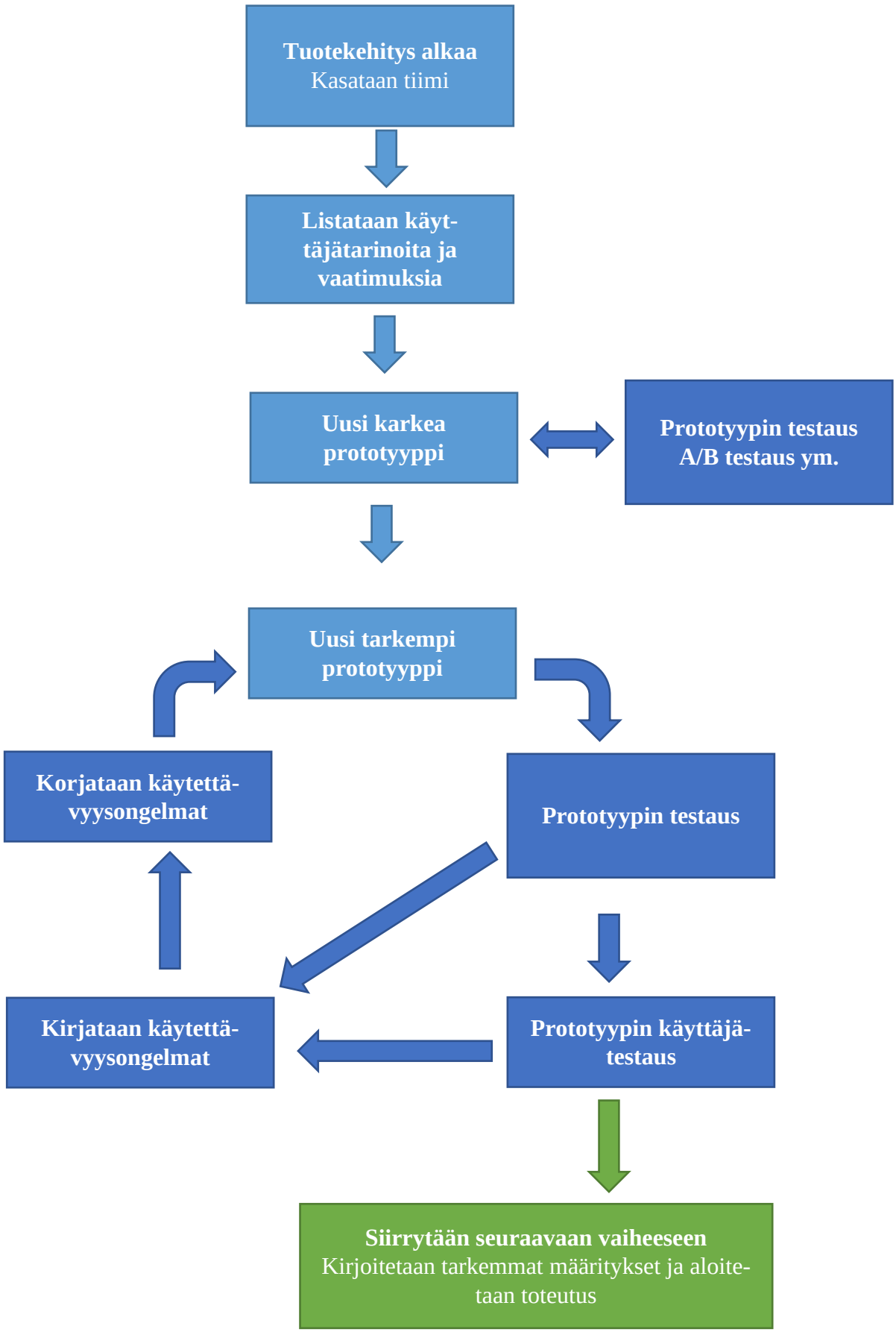
5 KÄYTÄNNÖN TYÖ: KÄSIKIRJA

Vaatimukset ja toimeksiannot tulevat Raisoftin asiakkailta eri asiakasvastaavien tai tuotteen omistajien kautta. Tämä käsikirja kattaa lähinnä niitä tilanteita, joissa suunnitellaan kokonaan uusi moduuli tai tehdään isompi refaktorointi olemassa olevalle moduulille.

Sprinttiin nostetaan sellaisia töitä, joista on olemassa käyttäjätarinoita ja joiden työmäärä voidaan arvioida. Tästä syystä käyttäjäkeskeistä työtä tekevän henkilön tai tiimin pitää olla vähintään 1–2 sprinttiä edellä muita.

Tämä käsikirja hyödyntää kaikkia yllä mainittuja teorioita, mutta projektista riippuen voidaan tilanteen mukaan arvioida, mitkä teoriat antavat parhaimman lopputuloksen.

Kuviossa 7 esitetään ehdotettu tuotekehityksen kulku. Projektin ei välttämättä tarvitse noudattaa tarkasti tätä kaavaa, vaan kaavaa sovelletaan siten, miten arvioidaan, että saadaan paras lopputulos.



KUVIO 7: Tuotekehityksen kulku.

5.1 Tuotekehityksen aloitus

Tehdään projektisuunnitelma ja kasataan tiimi. Tiimissä on hyvä olla vähintään tuotteen omistaja, tekninen asiantuntija (ohjelmistoarkkitehti tai kehittäjä) ja graafinen suunnittelija / prototyypin mallintaja. Tiimissä voi olla myös muita asiantuntijoita, kuten tekninen dokumentoija, domain-asiantuntija jne. riippuen projektin laadusta. Sama henkilö voi myös toimia useassa eri roolissa. Graafinen suunnittelija voi esimerkiksi olla myös prototyypin mallintaja.

Projektia pilkotaan mielellään pienempiin paloihin. Suunnitellaan pienin mahdollinen julkaisukelpoinen tuote (MVP = Minimum Viable Product). Tällä tavalla sen voi tuoda sprinttiin ja implementoinnin jälkeen julkaista asiakkaalle. Tekemällä tuote useassa pienemmässä erässä saadaan nopeammin palaute asiakkaalta/käyttäjältä ja voidaan hyödyntää tätä tietoa seuraavassa iteraatiossa.

5.2 Uusi prototyyppi

Prototyypin mallintaja suunnittelee prototyypin. Tässä työssä varmistetaan, että prototyyppi kattaa vaaditut käyttäjätarinat. Prototyypin mallintaja voi tehdä myös esimerkiksi A/B-vaihtoehtoja mikäli löydetään useampi mahdollinen ratkaisu. Prototyypin tulisi noudattaa samoja käyttöliittymäperiaatteita kuin muut ohjelmiston käyttöliittymät osat sekä noudattaa yleistä graafista ohjeistusta.

Valmista prototyyppiä voidaan esitellä eri sidosryhmille. Tässä halutaan varmistaa, että prototyyppi vastaa annettuihin käyttäjätarinoihin. Prototyypin palautteiden perusteella tehdään uusi prototyyppi, joka korjaa prototyypissä olevia puutteita. Tässä työssä täytyy olla tarkkana, ettei tuotteen alkuperäistä tarkoitusta muuteta tai ettei tuotetta ylisuunnitella.

5.3 Prototyypin käyttäjätestaus

On tärkeää, että loppukäyttäjät voivat testata prototyyppiä. Tähän on tärkeää valikoida sellaisia henkilöitä, jotka eivät ole olleet mukana suunnittelussa. Käyttäjätestauksessa käytetään yllä mainittuja teorioita. Esimerkiksi testattava henkilö saa tehtäviä, jota suorittaa. Testausta suorittava henkilö ei saa johdatella testissä.

Käyttäjätestauksen tuloksia käydään läpi ryhmässä. Arvioidaan mikä prototyypissä oli hyvää, ja mikä oli huonoa. Palautteiden perusteella voidaan tehdä uusi prototyyppi. Kun ollaan tyytyväisiä prototyyppiin, voidaan koota tarkempia määrityksiä.

5.4 Tarkemmat määritykset

Prototyyppi antaa tärkeää palautetta ohjelmiston käytettävyydestä. Ohjelmisto saattaa kuitenkin sisältää sellaisia ominaisuuksia, joita ei voida prototyypin avulla kerätä, esimerkiksi raportin laskentalogiikkaa tai hyväksymiskriteerejä. Näitä kriteereitä dokumentoidaan esimerkiksi käyttäjätarinan yhteyteen tai erilliseen dokumenttiin.

6 KOKEILU KÄSIKIRJAN AVULLA

Käsikirjaa kokeiltiin uuden ohjelmistomoduulin kehityksen yhteydessä.

6.1 Projektin aloitus

Projektia varten kasattiin tiimi, jossa oli mukana kolme asiakkuustiimin jäsentä. Nämä edustivat kolmea suurinta asiakasta. Näiden lisäksi mukana olivat ohjelmistoarkkitehti, graafinen suunnittelija, joka toimi myös prototyypin mallintajana sekä kaksi Raisoftin kouluttajaa.

6.2 Käyttäjätarinat

Listattiin kaikki tiedossa olevat käyttäjätarinat. Käyttäjätarinoita tehtiin kahden eri käyttäjätyypin näkökulmasta: normaali käyttäjä sekä pääkäyttäjä. Käyttäjätarinoita oli yhteensä 25. Näistä käyttäjätarinoista arvioitiin mitä voidaan jättää pois ensimmäisestä versiosta, jotta saataisiin ensimmäinen versio nopeammin toteutettua.

Käyttäjätarinat kerättiin suunnitteluvaiheessa ensin Word-tiedostoon. Word oli siinä mielessä hyvä, että käyttäjätarinoita oli helppo muuttaa suunnitteluvaiheessa, ja niitä saatiin lisäksi siististi kahdelle sivulle. Käyttäjätarinoista puuttui monesti *miksi* -kohta. Käyttäjätarinat määrittelivät siis käyttäjätyypin sekä tarpeen, mutta ei taustalla oleva syytä.

6.3 Liian yksityiskohtaiset suunnitelmat

Tarkoitus oli edetä niin, että löydetään karkeimmat viat mahdollisimman helposti ja nopeasti. Prosessi osoitti sen, että aluksi tehtiin liian yksityiskohtaiset suunnitelmat. Määritykset kirjoitettiin Word-tiedostoon, johon oli kirjoitettu myös käyttäjätarinat. Sivumäärä oli noin 15. Huomattiin, että tiimin jäsenet ja ulkopuoliset sidosryhmät eivät lukeneet määrityksiä niin tarkasti, että heillä olisi ollut tarkka kuva siitä, mitä oltiin tekemässä.

Kannattavampaa olisi ollut ensin tehdä karkea prototyyppi ja arvioida voidaanko sillä toteuttaa annettuja käyttäjätarinoita. Tämän jälkeen olisi vielä pitänyt tehdä tarkempi prototyyppi ja mahdollisesti vielä haastatella käyttäjiä ennen yksityiskohtaisten dokumentointien tekoa.

6.4 Prototyypin mallinnus

Tiimissä oli mukana graafinen suunnittelija, joka toimi myös prototyypin mallintajana. Prototyyppi mallinnettiin Adobe XD-ohjelmistolla. Mallinnus tehtiin sen jälkeen, kun oli tehty käyttäjätarinat ja määritykset. Tämä tehtiin liian myöhään, ja olisi pitänyt tehdä prototyyppi ennen määrityksien tekemistä.

Prototyypin käyttö oli erittäin onnistunut, sillä prototyyppi oli klikattava ja saatiin sen perusteella hyvä ymmärrys tiimin sisällä miltä tuote näyttäisi ja miten se toimisi. Prototyypin avulla löydettiin paljon puutteita ja niitä korjattiin sekä prototyypin seuraavissa versioissa että määrityksissä. Puutteet koostuivat enimmäkseen käyttäjäkokemusvirheistä eli sovellus ei ollut tarpeeksi helppokäyttöinen.

6.5 Heuristinen arviointi

Heuristinen arviointi on käytettävyysasiantuntijan arvio käytettävyydestä. Heuristinen arviointi suoritettiin ennen käyttäjätestausta. Koettiin, että tässä vaiheessa prototyyppi oli tarpeeksi hyvä, jotta heuristista arviointia voitaisiin tehdä. Heuristinen arviointi on helppo toteuttaa. Testauksen suoritti ohjelmistotestaaja. Heuristinen arviointi tuotti pieniä huomioita, kuten esimerkiksi puuttuvia ohjetekstejä.

6.6 Käyttäjähaastattelut

Haluttiin haastatella sellaisia ihmisiä, joilla ei ole ennestään kokemusta Raisoft-ohjelmasta. Käyttäjähaastatteluja varten tehtiin skripti, ja haastattelun suorittivat asiakkuustiimin jäsenet.

Haastateltiin yhteensä kolme henkilöä, aina yksi henkilö kerrallaan ja haastattelu tapahtui virtuaalisesti. Haastatteluun osallistuneet henkilöt työskentelevät sairaanhoitajina, jotka ovat Raisoftin kohderyhmää, mutta kellään heistä ei ollut aiempaa kokemusta tuotteestamme. Jokainen haastatteluun osallistunut sai tehtäviä, jotka heidän piti suorittaa. He saivat prototyypin käyttöönsä. Prototyyppi oli suunniteltu niin,

että tehtävät oli mahdollista suorittaa sen avulla. Haastatteleva henkilö pyrki olemaan hiljaa ja ohjaamatta haastatteluun osallistuvaa henkilöä. Haastateltavia kehotettiin myös ajattelemaan ääneen, jotta saadaan ymmärrys siitä, mikä on sovelluksessa vaikea ymmärtää.

Haastatteluiden avulla löydettiin selkeitä käyttäjäkokemusvirheitä. Käyttäjät eivät siis osanneet käyttää tiettyjä ominaisuuksia. Tämä tuli tiimille yllätyksenä, mutta osoitti samalla, että haastattelu toimii hyvin osana prosessia. Löydöksiä ei saatu esille sisäisissä testeissä, sillä testaajat olivat työstäneet prototyyppiä niin kauan, että olivat sokeita virheille.

Haastattelut olivat työläitä. Tulevaisuudessa käytetään todennäköisesti käyttäjähaastatteluja niissä tilanteissa, joissa suunnitellaan kokonaan uudestaan jokin moduuli tai laajempi kokonaisuus.

7 YHTEENVETO JA JOHTOPÄÄTÖKSET

Käsikirjaa ja menetelmiä kokeiltiin yhdessä projektissa, joka kesti noin kolme kuukautta. Projektissa suunniteltiin kokonaan uusi moduuli Raisoft.net-ohjelmistoon. Tämän työn yhteydessä kokeiltiin uusia menetelmiä, kuten käyttäjätarinoiden käyttämistä, pienin toimiva tuote (MVP), prototyyppien käyttämistä, käyttäjätestausta sekä heuristista arviointia.

Tuloksena saatiin, että käytettiin liikaa aikaa yksityiskohtaisten määrityksien tekemiseen alussa. Sen sijaan prototyyppien käyttö antoi nopeammin paremman tuloksen. Jatkossa uusia ominaisuuksia tehtäessä olisi syytä hyödyntää prototyyppejä enemmän.

Käyttäjähaastattelut toivat arvokasta palautetta. Niiden avulla löydettiin käytettävyysongelmia, joita ei ehkä voida laskea ohjelmistovirheiksi sinänsä, mutta joiden seurauksena käyttäjät eivät olisi osanneet käyttää ohjelmistoa oikealla tavalla. Haastattelut oli vaikeat järjestää ja suunnitella. Siitä huolimatta käyttäjätestaukset olisi hyvä järjestää jatkossa, kun suunnitellaan uusia ohjelmistomoduuleja.

Käytettävyysongelmia ei löydetty yrityksen sisäisissä kokeiluissa. Tämä johtuu ehkä siitä, että testaajat tiesivät etukäteen, miten ohjelman oli suunniteltu toimivan. Tästä syytä on tarpeen haastatella käyttäjiä, joilla ei ole ennestään kuvaa siitä, miten ohjelmiston on suunniteltu toimivan.

Aikaisemmin Raisoftilla ei ole käytetty käyttäjätarinoita. Työ on aloitettu suoraan, ilman käyttäjätarinoita, prototyyppejä tai haastatteluita. Paremman lopputuloksen aikaansaamiseksi olisi hyvä noudattaa tässä työssä esitettyjä menetelmiä.

Raisoft hyödyntää ketteriä kehitysmenetelmiä. Myös tässä työssä tuotettua käsikirjaa olisi syytä kehittää ketteriä kehitysmenetelmiä hyödyntäen. Käsikirjaa on hyvä päivittää sen mukaan, kun löydetään kehitysmenetelmiin parannuksia, jotka tuottavat paremman lopputuloksen.

LÄHTEET

Berteig, M. 2020. Scrum rules. Saatavissa: https://berteig.com/how-to-apply-agile/scrum-rules-each-sprint-results-in-a-potentially-releasable-product-increment/. Viitattu 12.5.2022

Brych, M. 2018. 7 Common mistakes when developing MVP. Saatavissa: https://espeo.eu/blog/7-common-mistakes-when-developing-mvp/. Viitattu 12.5.2022

Chapman, P. 2018. Exploring the Gestalt Principles of Design. Saatavissa: https://www.toptal.com/designers/ui/gestalt-principles-of-design Viitattu 12.5.2022

Fitt, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. PDF-dokumentti. Journal of Experimental Psychology, 47(6), 381–391. Saatavissa: https://doi.org/10.1037/h0055392. Viitattu 12.5.2022

Hick, W. E. 1952. On the rate of gain of information. Quarterly Journal of Experimental Psychology. Saatavissa: http://www2.psychology.uiowa.edu/faculty/mordkoff/InfoProc/pdfs/Hick%201952.pdf Viitattu 12.5.2022

Isherwood, M. 2018. How to write a user testing report. Saatavissa: https://uxdesign.cc/how-to-write-a-user-testing-report-that-people-will-actually-read-652d15d2f92e Viitattu 12.5.2022

Kartusek, J. 2020. How to Build an MVP In Agile. Saatavissa: https://www.cleevio.com/blog/how-to-build-an-mvp-in-agile/ Viitattu 12.5.2022

Ketola, M. 2022. SCRUM – Project model. Raisoft Intranet. Saatavissa https://rsoy.sharepoint.com/:p:/s/Developmentteam/EeBdDAHhRJBCtI_i8Fe6oUIB5G_urcFo6Em0FGhcE3yyg?e=yoTagY. Viitattu 12.5.2022.

Klimczak, E. 2013. Design for Software: A Playbook for Developers.

Koivumäki-Lindholm, M. 2013. Käyttäjätarinoilla ryhtiä asiakaslähtöisyyteen. Saatavissa: https://www.suomidigi.fi/blogit/kayttajatarinoilla-ryhtia-asiakaslahtoisyyteen. Viitattu 12.5.2022

Lam, R. 2020. How to create a user interview script. Saatavissa: https://medium.com/sketch-app-sources/how-to-create-a-user-interview-script-f0ee6ffd62b6. Viitattu 12.5.2022

Laubheimer, P, 2019. The 3-Click Rule for Navigation Is False. Nielsen Norman Group 11.8.2019. Saatavissa: https://www.nngroup.com/articles/3-click-rule/ Viitattu 12.5.2022

Luoma, T. 2018. Käytettävyyden hinta, kolme esimerkkiä. Saatavissa: https://handlaamo.fi/kaytettavyyden-hinta-kolme-esimerkkia. Viitattu 12.5.2022

Maryna Z. & Dmitry G. 2020. Clear Acceptance Criteria for User Stories with Examples. Saatavissa: https://rubygarage.org/blog/clear-acceptance-criteria-and-why-its-important. Viitattu 12.5.2022

McCracken. 2016. How to Conduct Usability Testing from Start to Finish. Saatavissa: https://uxmastery.com/beginners-guide-to-usability-testing/. Viitattu 12.5.2022

Miller, R. 2011. User interface design and implementation. Saatavissa: https://ocw.mit.edu/courses/6-831-user-interface-design-and-implementation-spring-2011/resources/mit6_831s11_lec17/. Viitattu 12.5.2022

Nielsen, J. 1992. Finding usability problems through heuristic evaluation. PDF-dokumentti. Saatavissa: https://course.ccs.neu.edu/is4300sp13/ssl/articles/p373-nielsen.pdf. Viitattu 12.5.2022

Nielsen, J. 1994. 10 Usability Heuristics for User Interface Design. Saatavissa: https://www.nngroup.com/articles/ten-usability-heuristics/. Viitattu 12.5.2022

Nielsen, J. 2000. Why You Only Need to Test with 5 Users. Saatavissa: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. Viitattu 12.5.2022

Nielsen, J. 2008. Interaction elasticity. Saatavissa: https://www.nngroup.com/artcles/interaction-elasticity/. Viitattu 12.5.2022

O'hEocha, C. & Conboy, K. 2010. The Role of the User Story Agile Practice in Innovation. Lean Enterprise Software and Systems - First International Conference, LESS 2010, Helsinki, Finland, October 17-20, 2010. Saatavissa: DOI: http://dx.doi.org/10.1007/978-3-642-16416-3_3. Viitattu 12.5.2022

Pichler, R. 2016. 10 tips for writing good user stories. Saatavissa: https://www.romanpichler.com/blog/10-tips-writing-good-user-stories. Viitattu 12.5.2022

Pokharel, A. Vaidya, P. 2020. Study of User Story in Practice. PDF-dokumentti. Saatavissa: http://dx.doi.org/10.1109/ICDABI51230.2020.9325670. Viitattu 12.5.2022

ProductPlan, How to Write an Epic (for Product Managers). Saatavissa: https://www.productplan.com/learn/how-to-write-an-epic/. Viitattu 12.5.2022

Raisoft. 2022. Www-sivut. Saatavissa: www.raisoft.com. Viitattu 12.5.2022

Rajeesh. 2018. Good UX is measured by ease of use, not by count of clicks. Saatavissa: https://www.appinessworld.com/blogs/177/Good-UX-is-measured-by-ease-of-use-not-by-count-of-clicks. Viitattu 12.5.2022

Rehkopf, M. 2022. Agile epics: definition, examples, and templates. Saatavissa: https://www.atlassian.com/agile/project-management/epics. Viitattu 12.5.2022

Schwaber, K & Sutherland, J. 2020. The Scrum Guide. PDF-dokumentti. Saatavissa: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf. Viitattu 12.5.2022

Schuurman, R. 2020. Product Owner vs Project Manager. Saatavissa: https://www.scrum.org/resources/blog/product-owner-vs-project-manager. Viitattu 12.5.2022

Tech Agilist. Definition of Done – Importance in Agile. Saatavissa: https://www.techagilist.com/agile/product-owner/definition-of-done-importance-in-agile/. Viitattu 12.5.2022

Usability. Reporting Usability Test Results. Saatavissa: https://www.usability.gov/how-to-and-tools/methods/reporting-usability-test-results.html. Viitattu 12.5.2022

Young, S. 2014. Improving Library User Experience with A/B Testing. Saatavissa: https://doi.org/10.3998/weave.12535642.0001.101. Viitattu 12.5.2022

ALOITUS

TERVETULOA+ESITTELYT:	Hei, Kiitos ja tervetuloa käytettävyystestiin. Loistavaa, että pääsit osallistumaan.
PUHELIN ÄÄNETTÖMÄLLE:	Pyydän, että laitat testin ajaksi **puhelimen äänettömälle**. Testissä kestää n. **tunti**.
TESTITILA+TALLENNUS:	Tää on nyt meidän käytettävyyslaboratorio ja itse **testitapahtuma tapahtuu tällä minun kannettavalla koneella.** Saisinko ottaa **jostakin virtaa**, ettei varmasti kone simahda ainakaan sen takia. Minun koneella on tallennusohjelma, jonka avulla **tallennan äänen, näyttökuvan ja naamakuvan**. Tallennuksen tarkoitus on se, että minun ei tarvitse tässä kirjoittaa kaikkia huomioita ylös vaan voin keskittyä tekemään sinun olon hyväksi ja huolehtia käytettävyyden analysoinnista sitten myöhemmin videoiden avulla. Pyydän sinua **hetken päästä** täyttämään **tallennuslupalomakkeen**. Minä säilytän tallenteen **yhden vuoden ajan** ja hävitän sitten. Tallennetta ei käytetä muuhun kuin **tuotteen kehittämiseen**.
SAA KYSYÄ:	Sää voit koska tahansa kysyä, jos sulla tulee mieleen jotain.

TESTIN ESITTELY JA TARKOITUS

KÄYTETTÄVYYSTESTI ON…:	Eli, me ollaan nyt tekemässä käytettävyystestiä ja **käytettävyystestejä tehdään siksi, että löydettäis järjestelmistä sellaisia ongelmia, joita suunnittelijoille ei tule mieleen.** Yleensä suunnittelijat on niin sisällä siinä asiassa, että niiltä jää väkisin asioita huomaamatta. Siksi on tosi **hienoa, että pääsit käyttäjänä tähän testiin mukaan,** koska sillä tavalla **saadaan paljon erittäin tärkeää tietoa.**
TALLENNUS:	Kuten kerroin, niin **testitilanne tallennetaan**, jotta käytettävyyttä voidaan analysoida tarkemmin jälkeenpäin.
TESTAAMISEN KOHDE:	Tässä testissä testataan **television prototyyppiä** ja tosiaan osallistumalla sää teet palveluksen kehittäjille, jotta löydetään parannettavia asioita palvelusta.
EI TESTATA KÄYTTÄJÄÄ:	Testissä **ei mitenkään testata sua** vaan testaamme tätä tuotetta. Sun rooli on tärkee, kun **oot täällä auttamassa meitä** tutkimaan tuotteen käytettävyyttä.

ONGELMIEN PAIKANNUS: Jos törmäät testin aikana ongelmiin, ei ole mitään syytä hämmentyä. **Ongelmakohtien paikallistaminen on juurikin se testaamisen tavoite**, jotta niitä voidaan sitten korjata ja tehdä tuotteesta helpompikäyttöinen.

TESTIN KULKU

TEHTÄVÄT: Tää testi etenee siten, että mää annan aina sulle **yksittäisiä tehtäviä**, joita sää koitat suorittaa tossa koneella. Tää ei oo mikään kilpailu, joten voit tehdä niitä ihan **omaan tahtiin** välittämättä siitä kauanko niissä kestää. **Välillä mää saatan puuttua asiaan kesken tehtävän** esimerkiksi auttamalla vähän tai voidaan siirtyä seuraavaan tehtävään, mutta **se ei oo mikään arvostelu** siitä, miten teet tehtävää vaan saattaa tarkottaa esim. sitä, että ollaan löydetty kohta, jossa tarvitaan kehittämistä palvelussa ja voidaan siirtyä eteenpäin.

HAASTATTELU: Kun tehtävät on tehty, mää vielä **haastattelen** sua hiukan.

KELLO: Mää seuraan **kelloa** niin, että sulla ei mee tähän koko hommaan kuin **maksimissaan tunti**.

SAA LOPETTAA: **Voit lopettaa testin** milloin tahansa ihan mistä tahansa syystä, eikä sun tarvitse selittää syytä, jos et halua. Samoin, jos joku yksittäinen tehtävä tuntuu ärsyttävältä etkä halua enää jatkaa sitä, **voit keskeyttää tehtävän** ja siirtyä seuraavaan tehtävään.

Onko sinulla tässä vaiheessa jotain kysyttävää?

ÄÄNEENAJATTELU

MITÄ AJATTELEE, EI TEKEE**:** Tässä tehtävien aikana mää **toivon, että sää ajattelet ääneen koko ajan**, koska me ollaan kiinnostuneita siitä, mitä sää ajattelet tehtävien aikana. Ääneenajattelu tarkoittaa sitä, että kertoo kaiken, **mitä ajattelee siitä lähtien, kun alottaa tehtävän** siihen asti, että se on valmis. Tärkeetä on nimenomaan kertoa, **mitä ajattelee, eikä mitä tekee.** Sen voi ajatella esim. niin, että olisi yksin koneella ja puhuisi itsekseen. Tärkeintä on jatkaa puhumista koko ajan. Koska se on luonnollista unohtaa välillä, niin **voi olla, että muistutan sua** aika ajoin jatkamaan puhumista.

ESIMERKKI: Mää annan lyhyen **esimerkin ääneenajattelusta.** Tässä on mun tehtävä, niin **anna mulle tää lappu** ja mää suoritan tehtävän ajatellen samalla ääneen:
Esimerkkitehtävänanto: Rakenna palapeli.

TEHTÄVÄNANTO LUETAAN:	Kuten huomasit, niin mää luin myös **tehtävänannon ääneen.** Tee sinä samoin, kun annan sulle tehtävälappuja. Sitten kun oot omasta mielestäsi **valmis tehtävässä, niin anna lappu takaisin mulle** merkiksi siitä. Tää on kans luonnollista unohtaa, niin **voi olla, että mää välillä muistutan sua** ilmaisemaan sen, että oot mielestäsi valmis. TARKKANA TÄSSÄ, ETTET OHJAA OSALLISTUJAA!
ÄÄNEENAJATTELUN HARJ.:	Mää annan nyt sulle yhden harjoitustehtävän, niin **kokeillaan tuota ääneenajattelua. Onko kysyttävää?**

ALOITETAAN TESTI

TEHTÄVÄNANNOT LUETAAN:	Muistutan vielä, että luet **tehtävänannot aina ääneen** ja ilmaiset, kun oot valmis. Sitten oli se **ääneenajattelu**, mistä saatan aika ajoin **muistuttaa**, että pitäis tehdä koko ajan.

TESTIN AIKANA
Laske aina **kymmeneen** ennen käyttäjän auttamista tai kehottamista.
Ilmaise että seuraat koko ajan, hymähdykset, nyökkäykset…
Anna **hyvää palautetta ääneenajattelusta**, jos on syytä.

Reagointi testitilanteessa:
- Ääneen ajattelusta muistuttaminen tarvittaessa. (Mitä ajattelet nyt, Muista ajatella ääneen)
- Jos käyttäjä on jumissa: Mitä yrität löytää – miksi haluat löytää sen – huomasitko tuon kohdan tuolla
- Pyydän sinua palaamaan sivulle X, mutta oli hyvä huomata, että tehtävän voi ajatella ratkaisevansa myös noin
- Jos osallistujalla on kysymys, johon ei voi vastata kesken testin – ota kysymys ylös ja käsittele haastatteluvaiheessa.
- Jos osallistuja tekee jotain, mihin haluat palata tehtävän jälkeen – kirjoita ylös ja esim. demonstroi koneella (”Teit tehtävässä X jotenkin näin – en silloin oikein kuullut, niin muistatko vielä, mitä ajattelit?)
- Jos tehtävä ei suju – pidä huoli, ettei osallistujalle jää paha mieli
- Jos tulee tekninen ongelma – pyydä osallistujalta lupaa päästä koneelle selvittämään asiaa