

Lappeenrannan teknillinen yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma

Kandidaatintyö

Niclas Lamponen

**OHJELMISTOTUOTANNON TILAUSTÖIDEN
ASIAKASTARPEISIIN VASTAAMINEN PIENYRITTÄJÄN
NÄKÖKULMASTA**

Työn tarkastaja(t): Tekniikan Tohtori Ari Happonen

Työn ohjaaja(t): Tekniikan Tohtori Ari Happonen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto

School of Engineering Science

Tietotekniikan koulutusohjelma

Niclas Lamponen 0455946

Ohjelmistotuotannon tilaustöiden asiakastarpeisiin vastaaminen pienyrittäjän näkökulmasta

Kandidaatintyö

2018

36 sivua, 4 kuvaa, 1 taulukko, 4 liitettä

Työn tarkastajat: Tekniikan Tohtori Ari Happonen

Hakusanat: pienyrittäjä, ohjelmistotuotanto, asiakastarpeisiin vastaaminen

Keywords: small business owner, software development, customer, response

Työn tavoitteena oli selvittää, miten pienyrittäjä pystyy parhaiten vastaamaan tilaustyön asiakastarpeisiin ohjelmistotuotannon kontekstissa. Työssä tutkittiin, miten pienyrittäjän ohjelmistotuotantoa voi parantaa, sekä mitkä johtavat projektien epäonnistumiseen. Kirjallisuuden tutkimisen lisäksi luotiin tilaustyönä ohjelmisto pienyrittäjän näkökulmasta. Työssä löydettiin syitä pienien projektien epäonnistumiseen: suunnittelun puute, matala prioriteetti, kokemattomat tiimit ja perinteisten projektinhallinta työkalujen käyttö. Työssä löydettiin myös pienyrittäjän asiakastarpeisiin vastaamisen kannalta oleelliset tekijät kuten pienyrittäjälle sopivimmat projektinhallinnan käytänteet. Sen lisäksi palautteen saaminen ja kommunikointi asiakkaan kanssa koettiin oleellisiksi pienyrittäjän asiakastarpeisiin vastaamisessa. Suunnittelun rooli alussa on merkittävä onnistumisen kannalta.

ABSTRACT

Lappeenranta University of Technology
School of Engineering Science
Degree Program in Computer Science

Niclas Lamponen 0455946

Responding to the customers' needs in software development from the perspective of small business owner

Bachelor's Thesis

36 pages, 4 figures, 1 table, 4 appendices

Examiners: D.Sc (Tech.) Ari Happonen

Keywords: small business owner, software development, customer, response

The goal of this thesis was to find out how small business owner could best respond to customers' needs in the context of software development. It was also examined how small business owners could improve their software development process, and which factors lead to the failure of projects. In addition to examining literature, a program was also developed from the perspective of small business owner. It was found that lack of planning, low priority, inexperienced teams and the usage of traditional project management tools are major factors for small project failure, whereas project management methodologies suitable for small business owner combined with feedback and communication to it were considered crucial in order to properly respond to customer needs. On top of that, proper planning and groundwork at the beginning is essential.

ALKUSANAT

Suuri kiitos Ari Happoselle kandidaatintyön ohjaamisesta.

SISÄLLYSLUETTELO

1	Johdanto.....	3
1.1	Tausta	3
1.2	Tavoitteet ja rajaukset	3
1.3	Työn rakenne.....	4
2	Projektinhallinta osana pienyrittäjän ohjelmistotuotantoa	5
2.1	Taustatyö ennen toteutusprojektin suorittamista.....	5
2.2	Projektinhallinnan erilaisten ketterien mallien soveltuvuus pienyrittäjän ohjelmistotuotantoon	6
2.3	Asiakasyhteistyön merkitys osana tilaustyötä.....	9
2.4	Tilaustöiden haasteellisuus pienyrittäjälle	10
3	Ohjelmisto	14
3.1	Ohjelmiston toteutus	14
3.2	Toteutukseen soveltuvien teknologioiden valitseminen	15
3.3	Kehitystyö	17
3.4	Käyttöönotto.....	18
4	Pohdinta ja tulevaisuus	20
4.1	Reflektio	20
4.2	Ohjelmiston jatkokehitys sekä ylläpito	22
4.3	Ohjelmistotuotannon menetelmien nykytilanne	23
5	Yhteenveto.....	24

SYMBOLI- JA LYHENNELUETTELO

CMMI	Capability Maturity Model Integration
CSS	Cascading Style Sheets
GPL	General Public License
HTML	Hypertext Markup language
HTTPS	Hypertext Transfer Protocol Secure
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
JS	JavaScript
LPGL	Lesser General Public License
MIT	Massachusetts Institute of Technology
MT	Megatavu
PK-Yritykset	Pienet ja keskisuuret yritykset
XP	Extreme Programming

1 JOHDANTO

Tässä luvussa käydään lävitse mitä tämä opinnäytetyö käsittelee sekä esitellään lyhyesti tätä opinnäytetyötä osana luotu ohjelmisto. Luvussa esitellään myös tämän opinnäytetyön tarkoitus, tavoitteet sekä rajaukset. Lopuksi esitellään koko opinnäytetyön rakenne.

1.1 Tausta

Tämä kandidaatintyö esittelee ohjelmistotuotantoa pienyrittäjän näkökulmasta. Työssä käsitellään mitä eri asioita pienyrittäjän tulisi ottaa huomioon ennen ohjelmistotuotannon prosessin aloittamista, sen aikana tai sen jälkeen. Työ esittelee sekä vertailee nykypäivänä pienyrittäjien keskuudessa suosittuja erilaisia menetelmiä sekä käytänteitä. Pienyrittäjän ohjelmistoprojektilla tässä kontekstissa tarkoitetaan yrittäjän ja mahdollisesti hänen tiimin (1-9 henkilöä) saaneen ohjelmiston tilaustyön suunnittelua, luomista, julkaisua sekä ylläpitoa ohjelmiston elinkaaren läpi.

Tässä työssä myös esitellään ohjelmisto, joka on tehty Lappeenrannan teknillisen yliopiston konetekniikan osaston käyttöön. Ohjelmiston tarkoituksena on laskea erilaisten mekaanisten liitosten välisiä voimia sekä liitosten kestävyyttä eri tilanteissa ja eri materiaaleilla.

1.2 Tavoitteet ja rajaukset

Tämän kandidaatintyön tavoitteena on selvittää, mitä ohjelmistokehityksen menetelmiä pienyrittäjän tulisi suosia itsenäisessä kehityksessä, sekä mitä asioita pienyrittäjän tulisi varoa ohjelmistokehityksen aikana. Tarkoituksena on myös luoda toimiva ohjelmisto, jonka luomisprosessia peilataan sekä verrataan kirjallisuuden oppeihin.

Työn tarkoituksena ei ole luoda tarkkaa rutiininomaista menetelmää pienyrittäjän ohjelmistoprojektiin, vaan tarkoituksena on vertailla sekä tutkia erilaisia menetelmiä, joiden tulosten pitäisi auttaa pienyrittäjää valitsemaan osuvampi menetelmä asiakastarpeisiin vastaamiseen, sekä toimimaan tehokkaammin ohjelmistotuotantoprosessin aikana.

1.3 Työn rakenne

Toisessa luvussa esitellään ohjelmistotuotannon menetelmiä, mitkä liittyvät ohjelmistotekniikan projektinhallintaan sekä vertaillaan niiden soveltuvuutta pienyrittäjän toimintaan. luvussa esitellään myös mitkä tekijät ovat avainasemassa pienyrittäjän ohjelmistotuotantoprosessin aikana. Kolmannessa luvussa käydään läpi tilaustyönä teetettävän ohjelmiston tuottamista. Neljännessä luvussa reflektoidaan tehtyä työtä, onnistumisia sekä käydään läpi tehtyjen valintojen pätevyyttä yleisellä tasolla. Lopuksi yhteenvedossa käydään lävitse työ kokonaisuudessaan lyhykäisesti.

2 PROJEKTIHALLINTA OSANA PIENYRITTÄJÄN OHJELMISTOTUOTANTOA

Projektissa on kyse väliaikaisesta suorituksesta, jolle on määritelty alku, loppu, laajuus sekä resurssit. Projektinhallinta on tietojen, taitojen ja työkalujen käyttämistä prosessin suorittamiseksi sen rajoitteiden puitteissa (pmi.org, 2018). Ohjelmistokehityksessä on yleistä hankkeiden jakaminen osaprojekteihin kuten esitutkimus, määrittely sekä toteutus. Osaprojektien tarkoituksena on muun muassa vaihtoehtojen kartoitus, projektin lopputuloksen kuvaus avulla sekä itse projektin toteuttaminen (Haikala ja Mikkonen, 2011). Lee ja Yong (2013) toteaa, että pienyritykset tarvitsevat joustavuuden takia hiukan erilaisen lähestymistavan ohjelmistotuotantoon verrattuna isompiin yrityksiin. Tässä opinnäytetyössä tarkastellaan mitä erilainen lähestymistapa pitää sisällään. Pienyrittäjän ohjelmistotuotannon lisäksi tarkastellaan myös muita ohjelmistotuotannon projekteja ja tutkimuksia, tyypillisesti pieniä ja keskisuuria, joiden soveltavuus voi koskea pienyrittäjiä.

2.1 Taustatyö ennen toteutusprojektin suorittamista

Haikalan mukaan ennen projektin aloittamista suoritetaan usein esitutkimusta asiakkaan puolesta, jossa asiakas käy lävitse mitä kautta ohjelmisto tilataan. Esitutkimuksessa käydään myös lävitse riskejä sekä kannattavuuden tutkimista. Esitutkimusta seuraa määrittelyprojekti, jossa luodaan toiminnallista määrittelyä ja kuvataan lopputulosta sen avulla. Lopulta toteutusprojektissa suoritetaan tekninen toteutus, johon kuuluu teknistä määrittelyä ja yksityiskohtaista suunnittelua. Suunnittelua seuraa ohjelmointi sekä testaus. (Haikala ja Mikkonen, 2011). Toteutusprojektiin liittyvä taustatyö käytettävistä teknologioista on avainasemassa ohjelmistotuotannossa. Käytettyjen teknologioiden valinta voi vaikuttaa suoraan asiakkaan maksamaan hintaan: pienessä yrityksessä, etenkin jos tiimin taito on rajoittunut tiettyihin ohjelmointikieliin niin uusien teknologioiden opettelu on aikaa vievää sekä kallista. Tehdessä projektia asiakkaalle voi myös pienyrityksen asiakkaan maksukyky tulla mahdollisesti vastaan. Määrittelyprojekti sekä yksityiskohtainen suunnittelu auttaa valitsemaan oikeat teknologiat ohjelmiston toteutukseen kuten ohjelmistokehykset, jotka eivät rajoita ohjelmiston tulevaisuutta. Epäsopivien teknologioiden valitseminen hidastaa tai voi jopa pysäyttää kehityksen, kunnes siihen liittyvät epäkohdat kuten mahdolliset lisensointi- ja yhteensopivuusongelmat ovat korjattu.

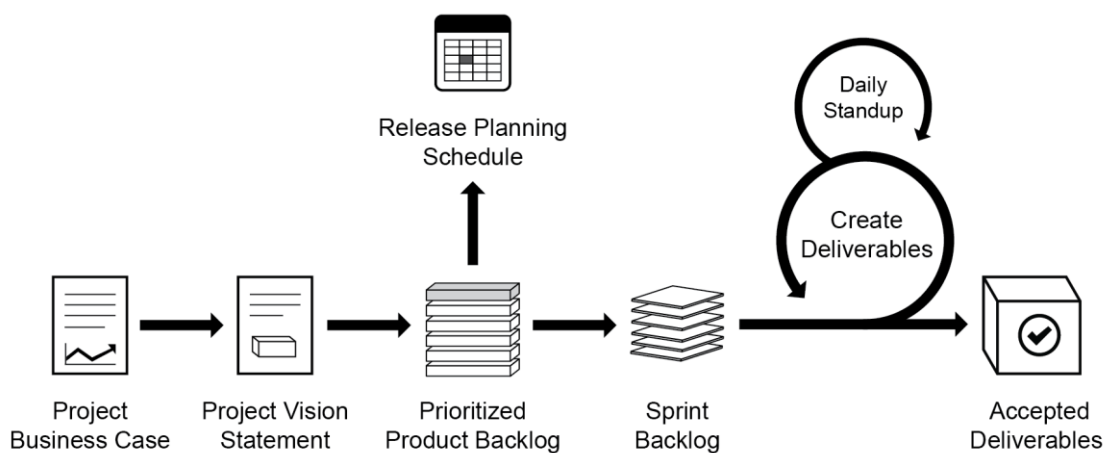
Taustatyöhön liittyy teknologioiden valitsemisen lisäksi myös projektinhallintaan liittyvät seikat kuten projektinhallinnan viitekehysten valinta.

2.2 Projektinhallinnan erilaisten ketterien mallien soveltuvuus pienyrittäjän ohjelmistotuotantoon

Viime vuosikymmenen aikana suositaan ovat keränneet etenkin ketterät menetelmät (Paetsch et al., 2003). Ketterillä ohjelmistokehityksillä tarkoitetaan ketterän ohjelmistokehityksen julistukseen (engl. ”Agile manifesto”) perustuvia ohjelmistotuotannon kehyksiä. Ketterä ohjelmistokehitys perustuu julistuksessa määritettyihin arvoihin sekä 12 erilaiseen periaatteeseen (Agilemanifesto.org, 2001). Erickson et al., (2005) kuvaavat ketterää ohjelmistotuotantoa seuraavasti: ”ketteryys tarkoittaa mahdollisimman suuren määrän raskauden poistamista, mikä usein yhdistetään perinteisiin ohjelmistotuotannon menetelmiin, jotta voidaan edistää nopeita muutoksia vaihtuviin ympäristöihin, muutoksia käyttäjävaatimuksiin, joudutettuihin määräaikoihin ja vastaavanlaisuuksiin.”. Tässäkin työssä keskitytään enemmän ketteriin menetelmiin, sillä Ericksonin määritelmän mukaan luonteeltaan ne ovat sopivampia pienyrittäjälle tapauksissa, joissa asiakkaan vaatimukset saattavat muuttua usein. Ketteristä menetelmistä tässä työssä puhuessa tarkoitetaan menetelmiä, jotka sopivat edellä mainittuun Erickson et al., määrittelyyn, sekä miten ne ovat ketterän ohjelmistokehityksen julistuksessa määritetty (Erickson et al., 2005; Agilemanifesto.org, 2001). Menetelmän valinta tulee kuitenkin suorittaa projektikohtaisesti.

Tutkimuksessa, jossa haastateltiin 98 eri yritystä selvisi, että ”yleisesti ottaen haastatellut olivat vakuuttuneempia ketterien menetelmien eduista kuin sen harvoista puutteista” (Vijayasarathy, L. ja Turk, D., 2008). Projektinhallinnan menetelmistä Scrum on tämän hetken suosituin ketterän ohjelmistokehityksen viitekehys (Rubin, 2012). Prosessi ei ole standardisoitu, vaan tarkoituksena onkin kehittää kehys, joka toimii kyseisessä projektissa. Tiimit koostuvat Scrum -tiimeistä, jotka yhdessä sidosryhmien kanssa luovat ohjelmistoa iteratiivisen prosessin avulla. Erittäin karkeasti esiteltynä iteratiivinen prosessi koostuu n. 2-4 viikkoa kestävästä sprinteistä (havainnollistettu kuvassa 1), joiden alussa käydään lävitse implementoitavat toiminnallisuudet, jotka aiotaan kehittää osaksi ohjelmistoa sprintin aikana. Sprintin aikana pyritään välttämään uusien muutosten vastaanottaminen. Sprinttiä seuraa sprintin läpikäyminen ja arviointi sekä ohjelmiston julkaiseminen. Scrum -tiimit

koostuvat tyypillisesti noin 5-10 hengestä. Scrum skaalautuu hyvin, sillä samassa projektissa voi olla useampi Scrum -tiimi työstämässä ohjelmistoa. Scrumille tyypillistä on myös päivittäiset tapaamiset (Rubin, 2012). Eräässä tutkimuksessa todettiin, että pienessä yrityksessä Scrum voi parantaa ohjelmiston tuottamisen prosessia ilman että laatu kärsii, ja sitä pidettiin hyvänä vaihtoehtona etenkin pienille yrityksille, joiden resurssit ovat rajoitetut. Scrumin joustavuus ja mukautuvuus projektin tilanteeseen ja kokoon sallii tiimien oman strategian laadunhallintaan, jolloin huomattiin, että laatu pysyi samana ja vaivannäkö pienenä (Caballero, et al, 2011).



Kuva 1. Scrum prosessi havainnollistettuna. (Satpathyt, 2014)

Kanban on lean-johtamisfilosofiaan perustuva projektinhallinnan viitekehys. Lean perustuu seitsemään periaatteeseen, joissa pyritään vähentämään hukkaa ja tuottamaan enemmän arvoa. Näiden lisäksi Kanbanilla on viisi omaa periaatetta (Ahmad et al., 2013), jotka ovat myös esitelty liitteessä 1. Kanbanin perustoiminta perustuu Kanban -tauluun, johon otetaan kehitysjonosta uusia ominaisuuksia työn alle, kun edelliset on suoritettu. Kanbanissa oleellista on työn alla olevan työn rajaaminen sopiviin määriin kehittäjiin suhteutettuna ja sen lisäksi Kanbanin erottaa Scrumista mm. roolien puuttuminen, visualisointi sekä jatkuva kehitystyö ja toimitus verrattuna Scrumiin (Kniberg et al., 2010). Kanbanissa ei ole rajoituksia tiimien kokoon sillä rooleja ei ole määritelty, eikä siinä tavata päivittäin tiimin kesken keskustelemaan tilanteesta. Amhad et al. (2013) toivat esille kirjallisuuskatsauksessaan, miten Kanban tarvitsee tuekseen muita käytänteitä, kuten muiden ketterien menetelmien yhdistämistä Kanbaniin. He mainitsivat, että Kanban pitäisi

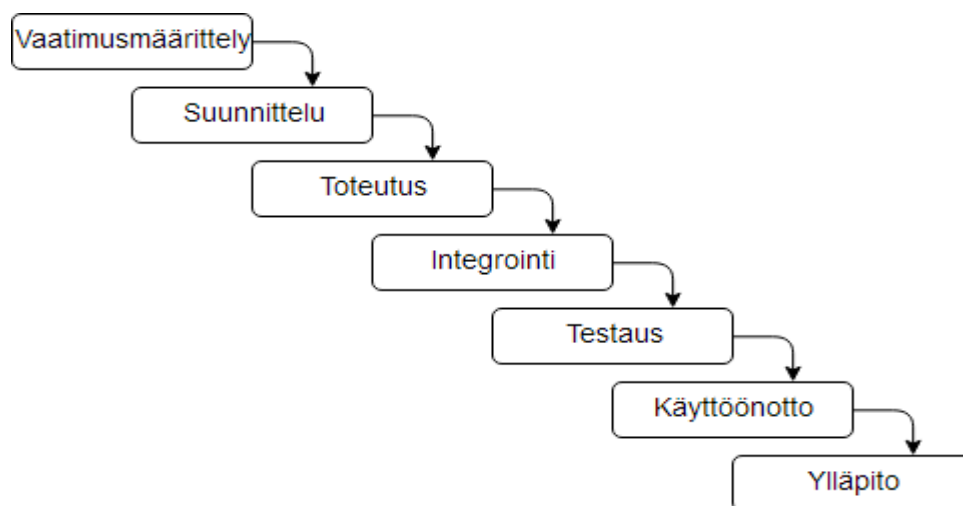
kytkeä olemassa oleviin metodeihin sen sijaan että se toimisi korvaajana toiselle projektinhallinnan viitekehykselle.

Ahmad et al. (2013) työssä raportoitiin, että monet ketterän ja Kanbanin yhdistävät tiimit kokivat, että se voi tuoda merkittävästi arvoa heidän yrityksilleen. Osa tiimeistä myös raportoi, että koki Kanbanin integroimisen vaikeaksi. Eräs suosituista on Scrumin yhdistäminen Kanbaniin. Scrumbanissa on tavoitteena ottaa Scrumin perusominaisuudet ja rakenne, mutta Kanbanin joustavuus sekä visualisointi (Brezočnik ja Majer, 2016; LeanKit, 2018). Scrumbanissa saatetaan käyttää iteraatioita kuten Scrumin sprinteissä. Mukaan on otettu Kanbanista työn alla olevien kohteiden rajoittaminen sekä visualisointi taulun avulla. Scrumbania pidetään Scrumia responsiivisempänä kehyksenä (Banijamali et al., 2017), joten muutokset kesken iteraation ovat hyväksyttäviä ja siten lisäävät joustavuutta. Tulevia kohteita kehitysjonosta priorisoidaan sen mukaan mikä tuottaa eniten arvoa asiakkaalle (Brezočnik ja Majer, 2016). Yhdistämällä molemmat saadaan siis joustava tarvittaessa iteratiivinen prosessi, joka seuraa ketterän- sekä lean-kehityksen periaatteita. Rooleja ei ole aina tarpeellista määrätä erikseen, mutta Scrumban kannustaa päivittäisiin tapaamisiin (Brezočnik ja Majer, 2016).

Taulukko 1. Kanbanin, Scrumin ja Scrumbanin erot. (Banijamali et al., 2017), käännetty lähteestä.

Kanban	Scrum	Scrumban
Ei ennalta määrättyjä rooleja	Ennalta määrättyt roolit	Ennalta määrätty roolit, saattavat vaihtua projektin
Jatkuva julkaisu	Aikarajojen määrittämät sprintit	Tauluun pohjautuvat iteraatiot
Keskeneräisen työn määrä rajoitettu	Sprintit rajoittavat työn määrää	Keskeneräisen työn määrä rajoitettu
Muutoksia voidaan tehdä milloin vain	Ei muutoksia kesken sprintin	Muutokset sallittu kesken sprintin
Aikaisempi suunnittelu ja dokumentointi tarpeellista	Suunnittelu tehdään jokaisen sprintin jälkeen	Suunnitellaan tarvittaessa, myös sprinttien aikana
Kanban taulu on pysyvä	Scrum taulu tyhjennetään sprintin lopussa	Scrumban taulu on pysyvä
Tehtävän kokoa ei ole rajoitettu	Tehtävän koko on rajoitettu sprinttiin	Tehtävän kokoa ei ole rajoitettu
”veto”-pohjainen työnhallinta [uusia tehtäviä ”vedetään” taululta]	Sprintin kehitysjonoon perustuva työnhallinta	”veto”-pohjainen työnhallinta [uusia tehtäviä ”vedetään” taululta]

Vesiputousmalli (kuva 2) on malli, jota pidetään usein perinteisenä ohjelmistotuotannon mallina. Vesiputousmallin iteratiivinen ominaisuus on tyypillinen ominaisuus mikä on peritty esimerkiksi Scrumiin. Muuten vesiputousmallia pidetään kuitenkin pitemmän ajan mallina, jossa on tiukat ohjeet projektin hallinnan ja roolien suhteen. Pyritään välttymään muutoksista kesken projektin ja asiakas on osallisena pääosin vain alussa (Fair, 2012). Korkala (2015) korostaa, että perinteisessä kehityksessä kommunikaatio asiakkaan kanssa painottuu usein selvästi dokumentoituun tietoon sekä se on luonteeltaan usein yksisuuntaista. Sama tutkimus tuo esille, että esimerkiksi ketterien lähestymistapa kommunikointiin on hiukan epävirallisempi ja pyritään tekemään kasvokkain. Epävirallinen kommunikointi saavutetaan tuomalla projektin sidosryhmät yhteen, luomalla luottamusta ryhmien välille.



Kuva 2. Vesiputousmalli

2.3 Asiakasyhteistyön merkitys osana tilaustyötä

Ketterän ohjelmistokehityksen julistuksen yksi neljästä arvosta on asiakasyhteistyön suurempi arvostaminen kuin sopimusneuvottelun (Agilemanifesto.org, 2001). Asiakastapaamisten merkitys korostunee etenkin ketterissä menetelmissä, joissa asiakkaan on usein tarkoitus olla mukana aktiivisesti koko kehitysprosessin ajan. Tutkimusten mukaan projekteissa, joissa asiakkaalle lähetetään usein rajoittuneen toiminnallisuuden omaavia

versioita tuotteesta kesken kehitysprosessin, menestytettiin paremmin. (MacCormack et al., 2001).

Asiakkaat ovat tärkeitä osa ennen projektin aloittamista vaatimusmäärittelyn aikaan, mutta ketterät menetelmät kehottavat asiakkaan osallistumista myös koko projektin läpi. (Paetsch et al., 2003). Eräässä tutkimuksessa asiakkaan osallistumista ohjelmistotuotannon projektiin pidettiin tärkeimpänä onnistumisen tekijänä (The Standish Group International, Inc, 1995). Korkalan (2015) mukaan palaute asiakkaalta tulee parhaiten fyysisessä yhteistyössä. Boehm et al. (2002) mukaan ketterät menetelmät toimivat parhaiten, kun asiakkaat toimivat omistautuneesti yhteistyössä kehittäjätiimin kanssa. Heidän mukaan asiakkaiden pitää olla ”omistautuneita, tietäviä, yhteistyöhaluisia, edustuksellisia ja valtuutettuja.”. Ominaisuudet eivät välttämättä ole rajoittuneet vain ketteriin menetelmiin sillä yleisesti ottaen asiakkaan omistautuneisuus projektille on koettu nostavan onnistumista. Korkala (2015) tuo esille myös kommunikaation tärkeyden, mutta huomioi myös sen, että kommunikaatio tiimin tai asiakkaan kanssa on pois kehitysjajasta. Eräässä toisessa tutkimuksessa puolestaan päädyttiin lopputulokseen, että paras tapa parantaa ohjelmistotuotannon projektin tuottavuutta ja laatua on keskittymällä ihmisiin. Tutkimuksen mukaan se tukee inkrementaalista ja iteratiivista kehitystä ohjelmiston pienissä julkaisuissa, sekä siten keskittyy kustannusten vähentämiseen ja laadun parantamiseen (Lee ja Yong, 2013). Tutkimuksissa tuotiin esille myös kommunikoinnin haasteellisuus sillä vastakkainen osapuoli ei välttämättä puhu samaa kieltä (van Waardenburg and van Vliet, 2013) ja että osapuolet saattavat tulla erilaisista sosioekonomisista taustoista, jolloin se hidasti heidän keskeistä kommunikointia (Banijamali et al., 2017). Banijamali et al. tutkimus esitti myös, että kommunikointiongelmien saattavat liittyä tehokkaiden kommunikointimekanismien puuttumiseen, mutta toi myös esille sen, että Scrumban johtaa hyvään määrään kommunikointia. Esimerkki asiakkaan osallistumisesta kehitysprosessiin on mm. luomalla käyttäjätarinoita. Paetschin et al. (2003) mukaan käyttäjätarinat ovat asiakkaiden tarvitsevia ominaisuuksia, jotka tuovat arvoa asiakkaalle, jotka on tarkoitus implementoida osaksi ohjelmistoa.

2.4 Tilaustöiden haasteellisuus pienyrittäjälle

Pienyrittäjille on kehitetty standardeja kuten ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) 9001 laatujohtamisesta, sekä ISO/IEC 29110: "Lifecycle profiles for very small entities". Pino et al. (2007) mukaan on olemassa kasvava huoli siitä, että esimerkiksi ISO-standardit eivät ole helposti sovellettavissa pieniin yrityksiin. Niiden soveltamiseen vaadittu aika, raha sekä resurssit vaativat suuren panostuksen. Vaikka esimerkiksi ISO/IEC 29110 kompleksisuutta on pyritty vähentämään, tutkimuksessa huomioidaan, että silti "kompleksisuus voi olla este rajoittuneiden resurssien takia." (Pino et al., 2007). Myös Coleman ja O'Connor (2008) on tuonut esille, että etenkin pienet yritykset ovat varuillaan ISO standardeista ja niiden vaatimasta dokumentoinnin määrästä, esimerkiksi ISO 9000 vaatii yrityksiltä tiukasti kontrolloidut sekä laajat dokumentaatiot sertifikaatin saamiseksi, tehden niistä haastavia pienyrittäjille. Sama tutkimus tuo esille perinteisten konseptin, CMMI:n (Capability Maturity Model Integration) epäsopimattomuuden pienille yrityksille.

Charette (2005) on esittänyt omat syyt miksi ohjelmistotuotannon projektit usein epäonnistuvat: Epärealistiset tavoitteet; Epätarkat arviot tarvittavista resursseista; Heikosti määritellyt järjestelmävaatimukset; Huono tiedotus projektin statuksesta; Huono kommunikointi asiakkaan, käyttäjän ja kehittäjän välillä; kyvyttömyys hallita projektin monimutkaisuutta sekä huono projektinhallinta. Rowe (2015) puolestaan on esittänyt viisi pienissä projekteissa esiintyvää haastetta.

- Suunnittelun puute
- Matala prioriteetti
- Kokemattomat tiimit
- Projektipäällikön vastuu monessa asiassa
- Perinteisten projektinhallinnan työkalujen ja prosessien käyttö pienessä projektissa.
-

Rowe mainitsee, että pääosin suuren projektin epäonnistumiseen johtavat ongelmat ovat esillä samalla tavalla pienessä projektissa, mutta uniikkia pienille projekteille on projektipäällikön maineen kärsiminen epäonnistumisen kohdatessa. Toisaalta esille tuodaan miten projektinhallinnan käyttäminen jo pienissä projekteissa kehittää projektipäällikön kykyjä toimia isompia projekteja varten. Rowen epäilykset vahvistavat vain sen, mitä jo valmiiksi tunnetaan projektien epäonnistumisesta ja että haasteet eivät ole uniikkeja pienille projekteille. Colemanin ja O'Connorin (2008) tutkimuksessa mainitaan, että "ohjelmistoalan

yritysten, etenkin start-uppien pitää olla joustavia, luovia, dynaamisia sekä kykeneviä tuottamaan tuotteita nopeasti selviytyäkseen.”

(Vahaniitty, et al., 2010) argumentoivat, että pienet yritykset tarvitsevat portfolion hallintametodeja, kuten projektiportfolion (organisaation projektit, jotka tavoittelevat organisaation strategisia tavoitteita (Planisware, 2018)) tarkastuksia sekä esimerkiksi samanaikaisten projektien määrän rajoittamista yhdelle henkilölle, eli toisin sanoen henkilöresurssien selkeää allokointia. Richardson ja Von Wangenheim (2007) puolestaan tuo esille pienyritysten tarpeen implementoida versionhallinta tarpeeksi ajoissa.

Turnerin et al., (2012) mukaan mikroyrityksien projekteista yli puolet ovat lyhyempiä kuin kolme kuukautta. Projekteihin Turner et al. pitää pk-yrityksille (pienet- ja keskisuuret yritykset) tärkeimpinä työkaluina vaatimusten hallintaa sekä resurssien aikataulutusta. Seuraavaksi tärkeimmiksi ehdotetaan työnjakoa, tavoitesuunnittelua (milestone planning) sekä laadunhallintaa. Turnerin et al., (2009) mukaan pk-yritykset tarvitsevat myös projektinhallintaa ”innovatiivisuuden keskittyneeseen hallintaan, kasvun saavuttamiseen ja strategisten tavoitteiden saavuttamiseen”. Turner et. al. myös väittää, että suurella osalla pienistä yrityksistä on kehnot projektinhallinnan käytännöt. Ongelman avuksi ehdotetaan projektinhallinnan käytänteitä, jotka tarjoavat helppoa suunnittelua ja ohjausta, epävirallista arviointia sekä raportointia. Hallinnan käytänteiden pitää olla kaikkien käytettävissä organisaatiossa niin, että ne tukevat optimaalista päätöksentekemistä. Turnerin et al. mukaan pienet mikroyrityksen eivät usein käytä projektipäällikköjä, jolloin amatöörit tarvitsevat yksinkertaistetut ja ihmiskeskeiset käytänteet, joita mikroyritykset voivat soveltaa ja käyttää helposti. Turner et al. ehdottaakin epävirallisia kevyempiä versioita projektinhallintametodeista, jotka keskittyisivät olemaan vähemmän byrokraattisia ja enemmän ihmiskeskeisiä (Turner et al., 2012).

Tutkimuksen mukaan on selvää, että projektinhallinta on haasteellista pienyrityttäjien kohdalla, joilla ammattitaitoa ei ole (Turner et al., 2009). Vaikka heidän mukaan pk-yrityksillä olisi tarve, niin heillä ei usein ole palkattu erikseen projektimanagereita. Projektienhallintaa saattaa usein siis hoitaa henkilöt, joille projektinhallinta ei ole täysin tuttua. Erittäin pienissä ohjelmistotuotantoalan yrityksissä on myös huomioitavaa, että projektinhallinnan taidon puutteen lisäksi yrityksen pitää huolehtia ohjelmistojen laadusta

(mm. käyttöliittymät, palvelimet, ohjelmiston jakelu, ylläpito, testaus), muuttuvista asiakasvaatimuksista sekä esimerkiksi lisensointiin liittyvistä asioista. Myös projektien ylläpito saattaa kuormittaa etenkin pienyritystä pienen henkilömäärän takia. Pienyrityksen taustalle on siis saatava taitava tiimi, joka pystyy ottamaan asiakkaan osaksi kehitysprosessia ja jolta löytyy halua projektinhallintaan sekä sen kehittämiseen.

3 OHJELMISTO

Tässä opinnäytetyössä testataan aiemmin kirjallisuudesta esille nostettuja väittämiä sekä ohjeita liittyen pienyrittäjän ohjelmistotuotantoon. Kokeilun konkreettisena tutkimuskohteena toimii Lappeenrannan teknillisen yliopiston konetekniikan osastolta tullut tilaus ohjelmistolle. Ohjelmiston toteutus suoritetaan pienyrittäjän näkökulmasta, simuloiden pienyrittäjän toimintaympäristöä. Heti projektin alussa tuli esille pienyrittäjän projektilla ominaiset muuttuvat vaatimukset, kun huomattiin mahdollisuus ohjelmiston kaupallistamiselle. Ohjelmiston piti alun perin tulla osaston sisäiseen käyttöön, mutta pian mahdollisuus kaupallistamiselle nostettiin esille ja alettiin harkitsemaan lisenssipohjaista liiketoimintamallia ohjelmistolle. Liiketoimintamallissa kolmannet osapuolet ostaisivat lisenssin ja saisivat sitä vastaan kopion ohjelmistosta. Kaupallistamismallista sekä markkinoinnista vastaa kuitenkin konetekniikan puolen tutkijat sekä professorit.

3.1 Ohjelmiston toteutus

Ohjelmistossa käyttäjälle oleellinen perustoiminto on materiaalin valitseminen, liitoksen tyyppin valitseminen sekä ominaisten lukujen, kuten materiaalin paksuuden, syöttäminen. Ohjelma laskee reaaliajassa käyttäjälle liitoksen kestävyysliittyviä avainlukuja, sekä kokoaa liitoksen tiedoista taulukon johon käyttäjä voi tarkentaa tietoja ja lopulta tulostaa tai viedä ohjelmasta johonkin toiseen ohjelmaan.

Ohjelmiston tuottaminen oli myös pitkälti sisäiseen käyttöön nykyisten tutkijoiden ja opiskelijoiden avuksi. Shoren ja Wardenin (2008), mukaan kyseinen kehitys sisäiseen käyttöön eroaa hiukan perinteisestä, sillä sisäisiin projekteihin palvelun kehittäjän pitää palvella sekä tuotteen tilaajaa että loppukäyttäjää, ja heidän tavoitteet saattavat olla erilaisia. Teoksessa kuitenkin todetaan myös, että hyvinä puolina on helposti saatavilla olevat loppukäyttäjät. Tämä tuli vahvasti esille myös tämän ohjelmiston kehityksessä, sillä pystyin olemaan suoraan kahteen loppukäyttäjään yhteydessä ja saamaan nopeasti erittäin tärkeää palautetta ohjelmiston kehittämistä varten.

Ohjelmiston kehitykseen projektinhallinnan osalta käytettiin aiemmin mainittua Scrumban menetelmää, jossa on yhdistetty ketterä menetelmä sekä Lean-ajattelun pääpiirteet, ottaen

erilaiset piirteet sekä Scrumista että Kanbanista. Työssä käytettiin Scrumbanista hiukan muokattua versiota, sillä ohjelmistoprojekti oli yhden henkilön toteuttama ja vain hyvin harva ohjelmistoprojektinhallinnan metodi on tarkoitettu vain yhden henkilön toteutettavaksi. Scrumbanin avulla saatiin ketterä sekä joustava projektinhallinta kehys, joka oli tarvittavan kevyt sekä tehokas projektiin, jossa ei ollut selvää etenemissuunnitelmaa ja vaatimukset sekä uusien ominaisuuksien priorisointi muuttui kuukausittaisten tapaamisten jälkeen. Ohjelmistoon lisättäviä toimintoja saattoi tulla jatkuvasti lisää ja ne saattoivat olla asioita, joiden implementointi piti priorisoida. Projektinhallinnan keveydellä tarkoitetaan, että esimerkiksi turhat tapaamiset, byrokratia sekä projektinhallintaan liittyvät asiat eivät vieneet aikaa liikaa kehitystyöltä.

Scrumbanissa työn alla olevasta työstä sekä tehdystä työstä pidettiin kirjaa Emacsin org-moden avulla perinteisen taulun sijaan, sillä tiedon ei tarvinnut olla esillä muille, mutta se oli kuitenkin tarvittaessa esitettävissä sidosryhmille sähköisenä. Projektin Scrumban taulu lokakuun puolelta esitetty kuvassa 2. Scrumin iteraatioita ei pidetty tiukasti ennalta määrättyinä ajankohtina, vaan niitä pidettiin noin kolmen tai neljän viikon pituisina. Iteraation lopussa työn tilaajat näkivät kehityksen ja pystyivät tarvittaessa ajoissa huomauttamaan korjausehdotuksia. Tämän lisäksi työn tilaajiin oltiin jatkuvasti yhteydessä sähköpostitse. Joustavan ketterän menetelmän avulla tilaustyön muuttuviin asiakastarpeisiin vastaaminen oli helppoa sekä tärkeimmät pikaisesti tarvittavat ominaisuudet pystyttiin priorisoimaan.

1	2	3
4	5	6
7 Test c++ modules	8 Recreate fatigue design window	9 C++ module linking
10 Create sheets for other joints	11 Remove PureCSS	12 Port calculations to C++
13 Create static design		14 Port calculations to JS for review DL 17.10
15 Add icons		16 Dynamically created material selection DL 17.10
17 New datastructure for materials		18 IPC communications between windows
		19 Main window & basic layout
		20 Background research on compatible frameworks
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
65		
66		
67		
68		
69		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80		
81		
82		
83		
84		
85		
86		
87		
88		
89		
90		
91		
92		
93		
94		
95		
96		
97		
98		
99		
100		

Kuva 3. Scrumban taulu projektin kehityksestä

3.2 Toteutukseen soveltuvien teknologioiden valitseminen

Tapaamisissa selvisi, että ohjelmiston tilaajilla ei ollut preferenssiä käytetystä teknologiasta. Ohjelmiston aiempi versio oli tukenut vain Windows -käyttöjärjestelmää, mutta mahdolliseen kaupallistamiseen liittyvistä syistä oli oleellista saada ns. “cross-platform”-ohjelmisto, eli ohjelmiston tuli toimia Windowsilla, MacOS:lla, sekä Linux-ytimeen pohjautuvissa käyttöjärjestelmissä. Tavoitteena oli myös saada tehtyä ohjelmisto, jonka kehittäminen sekä ylläpitäminen onnistuisi myös ulkopuoliselta henkilöltä kenellä on perusymmärrys web-ohjelmoinnista. Ohjelmistokehys (software framework) valittiin sen perustella, että se sopii myös projektin ulkopuoliselle opiskelijalle, joka on suorittanut WWW-sovellukset kurssin. Näin projektia voi tarvittaessa jatkaa toinen henkilö myöhemmin ohjelmiston päivityksiin tai ylläpitoon liittyvissä asioissa. Lisäksi tarvittiin ohjelmistokehys minkä lisensointi salli kaupallistamisen. Alustava vaatimusmäärittely jätti vielä paljon auki, jonka takia ohjelmistoa alettiin kehittämään ketterällä menetelmällä, mikä mahdollisti mukautumisen nopeaan vaatimusten vaihtumiseen.

Alustavaa palaveria seurasi välittömästi sopivan ohjelmointikielen valinta sekä siihen liittyvien kirjastojen tai ohjelmistokehysten valinta. Tässä vaiheessa taustatutkimuksen huolellinen tekeminen on ohjelmistotuotantoprosessin kannalta äärimmäisen tärkeää: väärin tekniikoiden valinnat tässä vaiheessa koituvat kalliimmiksi mitä enemmän ohjelmistoa kehittää niillä. Teknologioita selviteltiin tutkimalla sekä kyselemällä kokeneemmilta kehittäjiltä. Alustavasti vaihtoehtoina olisi mahdollisesti toiminut Electron, NW.js, tai Python/C++ graafisen käyttöliittymäkirjaston Qt:n kanssa. Qt:n lisensoidut kirjastot/ohjelmistokehykset ovat kuitenkin lisenssin GPL-3 (GNU Public License v3) tai LGPL-3 (GNU Lesser General Public License v3) alla, joka tarkoittaa, että lähdekoodista on tehtävä julkista 3 vuoden ajaksi LGPL:n alla tai pysyvästi mikäli kyseessä on GPL-3 lisensoitu ohjelmisto. Selvityksen jälkeen Electron tarjosi parhaan ympäristön, joka soveltui projektin luonteeseen täyttäen vaatimukset.

Lopulta MIT-lisensoitu (Massachusetts Institute of Technology) Electron valittiin ohjelmistokehykseksi ohjelmistolle. Kyseinen ohjelmistokehys sallii työpöytäsovellusten luomisen kaikille edellä mainituille käyttöjärjestelmille samalla koodilla. Myös jokaiselle käyttöjärjestelmälle erikseen pakkaaminen, eli ohjelman sekä tarvittavien kirjastojen liittäminen sekä suoritettavan ohjelmatiedostomuodon luominen onnistuu yhdellä komennolla. Kehittämisestä helppoa sekä melko nopeaa tekee se, että Electronilla luodut

ohjelmat tehdään erittäin yleisillä ja nopeasti opettavilla kielillä. Kehittäjän tarvitsee vain ymmärtää webkehityksen perusteet, kuten Node.js:n, JavaScriptin, HTML:n (HyperText Markup Language) sekä CSS:n (Cascading Style Sheets) perustiedot aloittaakseen kehittämisen Electronilla. Electronin tueksi käyttöliittymään otettiin käyttöön Materialize.css CSS-kehys modernin näköisen käyttöliittymän tekemiseen.

3.3 Kehitystyö

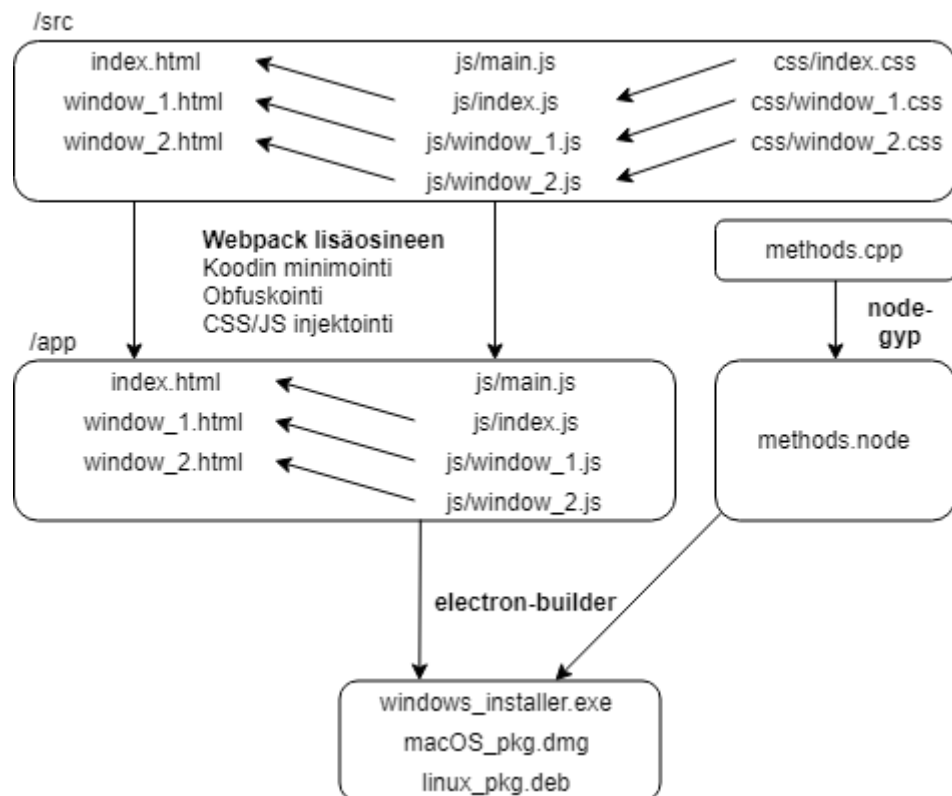
Kehitystyö aloitettiin välittömästi syyskuun alussa, sillä ensimmäinen soveltuvuusselvitys (Proof Of Concept), haluttiin valmiiksi lokakuun puoliväliin mennessä katsaukseen, jossa käytiin läpi mahdollisia investointeja kehitystyöhön. Tavoitteen saavuttamiseksi ketterää menetelmää sovellettiin vapaaluontoisemmin, jotta tavoitteet tapaamiseen täyttyvät. Etenkin tämänkaltaisissa tilanteissa kuitenkin Scrumbanin valinta kehitystyöhön vaikutti parhaalta, sillä asiakasvaatimuksiin vastaaminen oli merkittävästi kätevämpää sen joustavuuden ansiosta. Muutosehdotuksia pystyttiin vastaanottamaan kesken sprinttien sekä ne pystyttiin priorisoimaan, jotta aikataulut tiettyjen ominaisuuksien kohdilla saavutetaan. Myös tässä tilanteessa priorisoitiin esimerkiksi suorituspolun demonstrointi turvallisuuden kustannuksella, ja lopulta suorituspolku myös valmistui ennen tapaamista.

Ohjelmistoa kehitettiin läheisessä yhteistyössä konetekniikan tutkijoiden sekä professorien kanssa. Ohjelmistoa kehitettiin pääosin 5.5 tuntia päivässä 11.09.2018 alkaen, joten ohjelmiston kehitys eteni reipasta tahtia. Asiakkaan puolelta teetettiin hahmotelmia, joissa havainnollistettiin ohjelmiston toimintaa, selvennettiin käyttöliittymän tarpeita sekä ikkunoiden välisiä suhteita, jolloin itse keskityin ohjelmoinnilliseen toimintaan. Asiakkaan omistautuneisuus projektille sekä tietämys projektin aiheesta helpotti merkittävästi kehitystä osaltani.

Electronilla kehittäessä suurin huoli oli lähdekoodin helppo selvittäminen. Electron, joka tässä tapauksessa toimii osana Node.js:n ekosysteemiä, sallii Node.js:n avulla C++:n binääritiedostojen linkittämisen osaksi suoritusta, jolloin koodi pysyy paremmin suojassa ulkopuolisilta. Electron-builderin ja node-gyp:in avulla linkittäminen tapahtuu luomalla binding.gyp tiedosto projektin juureen sekä rakentamalla C++ binääritiedostot komennolla ”node-gyp rebuild”. Muita syitä C++ moduulien käyttämiseen on muun muassa joissain

tapauksissa koodin tehokkuus ja C++ rajapinnan tuomat edut kuten monet matalan tason kirjastot, jotka sallivat pääsyn tietojärjestelmään. (Hinkelmann, 2017; Nodejs.org, 2018)

Kehitystyö itsessään oli suoraviivaista. Projektinhallinnan malli seurasi aiemmin mainittua Scrumbanin mallia, mutta tiimin sijasta kehitin ohjelmistoa itse. Javascript, HTML ja CSS moduulit linkittävät yhteen Node.js:n paketti Webpack lisäosineen. Webpack lukee lähdekoodikansion sisään ja tuottaa lisäosien avulla kansion, jonne lopulliset moduulit rakentuvat. Webpack lisäosineen hoitaa mm. JavaScript koodin obfuskoinnin, minimoinnin, sekä siivoamisen kommentteista. Ne linkittivät tarvittavat moduulit keskenään sekä injektioi tarpeelliset CSS -tyylit osaksi .html tiedostoja. Lisää moduulien välisten suhteiden havainnollistamisesta löytyy kuvasta 3. Lopulta electron-builder voi lukea Webpackin luoman kansion ja rakentaa siitä jokaiselle käyttöjärjestelmälle lopullisen asentajan ohjelmistolle. Ohjelmiston avausikkunaa on esitelty sen alpha vaiheessa liitteessä 4.



Kuva 4. Suunnitellun ohjelmiston moduulien väliset suhteet kehitysvaiheessa.

3.4 Käyttöönotto

Paketointiin Electron tarjoaa useita eri vaihtoehtoja Node.js pakettien muodossa kuten electron-packager, electron-forge tai electron-builder. Ohjelmalle voi myös tarvittaessa tehdä Windows asentajan electron-builderin avulla. Tämän ohjelman paketointiin käytetään juurikin electron-builderia, jonka etuna on kaikkien käyttöjärjestelmien pakettien luominen yhdellä komennolla sekä automaattisten päivitysten tukeminen. Toistaiseksi ohjelmiston jakelu on tapahtunut vain tietyille sidosryhmille yleisten tiedostojen jakopalveluiden kautta.

4 POHDINTA JA TULEVAISUUS

Kirjallisuudesta saatujen oppien mukaan luodusta ohjelmistosta sai mielenkiintoisen työn aiheen. Tässä luvussa käydään läpi vielä reflektio ohjelmiston tuottamisesta ja tehdyistä valinnoista ottaen huomioon projektinhallinnan, taustatyön, ohjelmoinnin sekä tulevaisuudennäkymät. Lisäksi tehdään pikainen katsaus ohjelmistotuotannon menetelmien nykyhetkeen sekä tulevaisuuteen.

4.1 Reflektio

Aiemmin työssä esitettiin Rowen (2015) esittämät viisi haastetta pieneen projektiin:

- Suunnittelun puute
- Matala prioriteetti
- Kokemattomat tiimit
- Projektipäällikön vastuu monessa asiassa
- Perinteisten projektinhallinnan työkalujen ja prosessien käyttö pienessä projektissa.

Vastasin haasteisiin selkeästi panostamalla suunnitteluun, jolloin huonoin tapaus (kykenemättömyys tuottaa ohjelmistoa) selviäisi mahdollisimman pian. Taustasuunnittelun aloitin jo hyvissä ajoin, ja pyrin valitsemaan nykystandardien mukaan modernit sekä ylläpidettävät ratkaisut teknologioiden puolelta. Prioriteetti ei ollut ongelma, sillä olin yksin työskentelemässä ohjelmiston parissa joka päivä saman verran (mikä myös auttoi mm. tulevien arvioiden tekemistä kehityksen suhteen). Kokemattomuus tuli kuitenkin välillä vastaan. Yleensä tästä selvittiin kovalla googlailulla ja selvittelemisellä, mutta kokemattomuuden vaikutuksen ohjelmiston kehitysnopeuteen tiettyinä hetkinä huomasi. Koska simuloin pienyrittäjää, myös projektipäällikön vastuu oli osana työnkuvaa. Se ei tullut kuitenkaan esille mitenkään vahvasti tämän projektin aikana; pidin yhteyttä asiakkaaseen säännöllisin väliajoin, priorisoin sen mukaan tehtäviä ja olin vastuussa teknologiaan liittyvistä valinnoista. Mikäli olisin pienyrittäjä ja ympärillä olisi tiimi, pitäisin tiimin mukana em. asioissa, keskustellen asiasta heidän kanssaan ennen päätösten tekemistä. Projektin tuotos on kuitenkin aluksi vain sisäiseen kehitykseen ja osa isompaa kokonaisuutta mikä vaikutti sen luonteeseen. Asiakkaan kanssa kommunikointi oli melko vapaamuotoista.

Electronin valinta vaikutti hyvältä nopeaan kehitykseen, mutta siihenkin liittyvät haasteet tulivat pian esille. Electronin perustoimintamalli liittyy pääosin siihen, että ohjelmisto luo verkkoselain Chromiumin ikkunoita ja lataa niihin JS, HTML & CSS tiedostot, joten mallia voisi ajatella hiukan kuin internetsivuna. Puhtaat JS, HTML sekä CSS valittiin esimerkiksi Reactin tai Angularin kaltaisten kirjastojen sijaan siksi, että ohjelmasta tulee todennäköisesti todella pitkäikäinen. Nopeasti kehittyviltä JavaScript kirjastoilta pyrittiin välttymään, vaikka ohjelmisto onkin integroitu vahvasti Node.js:n ekosysteemiin. Työskentelen vain rajatun ajan projektin parissa, jonka jälkeen sitä tulee huoltamaan todennäköisesti toinen tietotekniikan opiskelija. Tarkoituksena oli, että hänen pitäisi olla helppo päästä osaksi projektia ja sisälle sen toimintaan sen sijaan että hän joutuisi työskentelemään vanhentuneen Angularin version ja syntaksin kanssa. HTML:n standardit kehittyvät huomattavasti vakaammin sekä pienemmin muutoksin verrattuna JavaScript kirjastoihin.

Suurimmat haasteet kaupallistetussa Electron ohjelmistossa tulee esille siinä, että käyttäjällä on silloin lähes täysi pääsy lähdekoodiin. JavaScript tiedostoa esimerkiksi ei käännetä ihmiselle lukukelvottomaksi binääritiedostoksi kuten C –kielen tiedostoja. Lähdekoodin tiedostoja voi koittaa yhdistää ”asar” -tiedostoon sekä muokata vaikeammaksi lukea, eli obfuskoida. Lopulta lähdekoodi on kuitenkin täysin käyttäjälle saatavissa sekä melko helposti luettavissa. Lopulta tähän sai paremman ratkaisun linkittämällä C++ binääritiedostoja osaksi Electron sovellusta, joita pystyi kutsumaan Node.js:n tarjoaman rajapinnan kautta. Binääritiedostojenkaan toiminta ei ole täysin piilotettu, sillä kaikki asiakkaalle lähetetty koodi on teoriassa mahdollista takaisinmallintaa. Tämänlaisen ohjelman tapauksessa kuitenkin siihen käytetty aika kävisi kalliimmaksi kuin esimerkiksi lisenssin ostaminen. Electronin kanssa samalla alueella toimiva NW.js tarjoaa Electronin kaltaisia ratkaisuja, perustuen samanlaiseen toimintamalliin. NW.js:n etuna on myös JavaScript lähdekoodin suojaaminen. Electronille tehdyt lisäosat julkaisemisen, päivittämisen sekä ylläpidon avuksi ovat syy, miksi se valittiin tähän NW.js:n sijaan.

Toinen pian esiintyvä potentiaalinen ongelma oli tiedostokokoo. Electronin mukana itsessään tulee paljon ominaisuuksia, jotka eivät välttämättä ole kaikille tarpeellisia. Jo pelkästään Chromium -selaimen sisällyttäminen jokaiseen paketoituun ohjelmistoon johtaa melko suureen tiedostokokoon. Pienimmillään ohjelmista tulee n. 50 Megatavun kokoisia, ja jokaiselle käyttöjärjestelmälle tulee oma erillinen tuon kokoinen paketti. Lopulta Electronin

valinta tuntui kuitenkin todella luontevalta tähän projektiin, sillä nykypäivänä ison binäärin liikutteleminen tai lataaminen on melko pieni ongelma, sekä tiedostokoon kustannuksella kehitystä saatiin nopeutettua.

Ohjelmaa luodessa myös tuvallisuuheen liittyvät kysymykset heräsivät mieleen. Electronin sivuilla oli kehittäjille ohje ohjelmiston turvaamisesta, mutta se ei kuitenkaan käynyt läpi Node.js:ään liittyviä riskejä. Electron on eräs Node.js ekosysteemin monista paketeista. Node.js:n kautta käyttäjä voi ladata monia muita paketteja eli esimerkiksi ohjelmistoja (paketti on hyvin monikäsitteinen käsite Node.js ympäristössä, joten tässä kontekstissa paketilla tarkoitetaan ohjelmistoa tai ohjelmaa, joka on osana luotavaa ohjelmistoa). Kun käyttäjä asentaa Electronin, on Electron riippuvainen Node.js:n monista muista paketeista kuten x, y ja z, jotka toisinaan ovat riippuvaisia muista paketeista a, b ja c. Vaikka tässä työssä esiteltyä ohjelmistoa varten oli asennettu tämän kirjoitushetkellä (28.11.2018) alle kymmenen pakettia, niiden riippuvuudet johtivat yli satojen pakettien asentamiseen. Tämä on johtanut kritiikkiin Node.js:n ekosysteemiä kohtaan (Beyer, 2018). Node.js:n toimintamalli paketteineen on hyvin standardi tietotekniikan puolella, mutta ongelmia syntyy, jos yksikin pakettiriippuvuuksista osana ketjua on haitallinen. Eräs viimeaikaisista esimerkeistä oli marraskuussa 2018 havaittu ”event-stream” nimiseen pakettiin ujutettu Bitcoinien varastaja, joka oli saavuttanut jatkuvasti yli miljoona latausta viikossa (GitHub, 2018).

4.2 Ohjelmiston jatkokehitys sekä ylläpito

Ohjelmistoon tulevista ominaisuuksista mainittiinkin jo hiukan. Kehitys jatkuu samalla kaavalla mikä on jo todettu toimivaksi. Perustoiminnallisuus on nyt melko hyvässä kunnossa, jotta alpha versiota ohjelmasta voidaan käyttää hyvin pian sisäisesti sen oikeaan tarkoitukseen. Ylläpidon puolelta ohjelmistosta on tarkoitus tehdä mahdollisimman ylläpidettävä, jolloin riippuvuudet esimerkiksi pian vanhentuviin paketteihin tai teknologioihin pudotetaan. Ylläpidettävällä ohjelmistolla tarkoitetaan dokumentoitua, selkeästi koodattua ja kommentoitua ohjelmistoa, jossa tulevan henkilön ei tarvitse käyttää esimerkiksi kirjastoa, joka on muutaman vuoden päästä vanhentunut tavalla mikä vaikeuttaisi kehitystyötä. Tämä voisi tarkoittaa esimerkiksi Angularin kaltaisen ohjelmistokehyksen välttelyä, jonka syntaksi saattaa kehittyä muutaman vuoden välein.

Ylläpidettävyyttä lisää myös Docker, joka huolehtii, että kehitysympäristö on identtinen myös muille kehittäjille ja ongelmilta ohjelmistojen eri versioiden välillä välttään.

Tulevaisuuden varalle ohjelmistoon aiotaan implementoida ohjelmiston jakelu sille suunnattujen palvelinten kautta, automaattiset päivitykset sekä koodin allekirjoitusvarmenteita. Electron-builder sisältää ”autoUpdater” nimisen moduulin, joka sallii koodin päivittämisen geneeriseltä HTTPS (Hypertext Transfer Protocol Secure) -serveriltä. Koodin allekirjoitusvarmenteet puolestaan tarkoittavat ohjelmiston digitaalista allekirjoittamista sen alkuperäisyyden varmentamiseksi. Lähitulevaisuudessa ohjelmaan myös tullaan mahdollisesti kaupallistamista varten implementoimaan lisenssipohjainen kaupallistamisen salliva mekaniikka, jossa käyttäjä ostaa avaimen ja aktivoi tuotteen sillä. Kyseessä on kuitenkin ohjelmiston kannalta viimeisimpiä päivityksiä mitä siihen tullaan tekemään. Tämän lisäksi lopulta tullaan myös valmistaman dokumentaatio ohjelmistosta. Ohjelmiston ylläpidettävyyys pitäisi näin onnistua ongelmitta myös pidemmällä aikavälillä.

4.3 Ohjelmistotuotannon menetelmien nykytilanne

Katsaus ohjelmistotuotannon menetelmien nykyhetkeen paljastaa ketterien menetelmien herättävää kiinnostusta. Ketterän ohjelmistokehitykseen liittyvät tieteelliset julkaisut ovat myös kasvaneet 2000 -luvun aikana vakaasti. Korkala kirjoittaa ketterien menetelmien kehityksestä kuten Leanin yhdistämisestä ja siitä että kyseinen trendi Leanin yhdistämisestä ketterään on mahdollisesti vielä kasvamassa. Korkala myös arvioi, että ketterän kehityksen menetelmien käyttökohteet sekä niiden laajennukset ovat vielä edelleen kehittymässä (Korkala, 2015).

Google Trends paljastaa myös hiukan ohjelmistokehityksen menetelmistä välillä 01/2014 – 11/2018 (esitelty liitteissä 2 ja 3). Hakutulokset liittyen ketterään ohjelmistokehitykseen ovat kasvaneet melko tasaisesti viimeisen vuosikymmenen aikana. Samalla kun perinteisten ohjelmistokehitys mallien kuten vesiputous tai XP (Extreme programming) suosio on laskenut, on Scrum selvästi herättänyt kiinnostusta. Datasta ei tule kuitenkaan vetää johtopäätöksiä sillä Google Trends on vain ”puolueeton otos Googlen hakudataa” (Rogers, 2016). Kuvaajat on luotu tueksi visualisoimaan Scrumin ja ketterien menetelmien herättämää kiinnostusta.

5 YHTEENVETO

Tässä työssä tukittiin miten pienyrittäjä voi parhaiten vastata asiakastarpeisiin, sekä miten pienyrittäjän ohjelmistotuotantoprosessia voi kehittää. Lopulta kirjallisuuden ohjenuorat mielessä pitäen luotiin ohjelmisto, jota seurasi reflektointi kirjalliseen osioon, käyden lävitse onnistumisia sekä haasteita. Tutkimusten johtopäätösten perusteella projektikohtaisesti tiettyjen kevyiden ketterien menetelmien hyödyntäminen pienyrittäjälle voi auttaa arvon lisäämisessä (Vijayasathy, L. ja Turk, D., 2008; Caballero, et al, 2011). Koska on tärkeää, että pienyrittäjät ovat joustavia (Coleman ja O'Connor, 2008), ketterä menetelmä lienee hyvä valinta, jotta asiakastarpeisiin pystytään vastaamaan nopeasti ja silti pitämällä byrokraattiset toiminnot minimissä. Näin tärkeimmät ominaisuudet priorisoidaan ja pystytään luomaan yhteys asiakkaaseen, pitäen kommunikointia yllä. Ketterän menetelmän valinta pienyrittäjälle on luontevaa myös sen takia, että asiakas on tarkoitus ottaa läheiseen yhteistyöhön mukaan projektiin, jolloin asiakkaalta saadaan palautetta usein. Sillä ketterät menetelmät on luotu vastaamaan muutokseen, voidaan palautteen pohjalta saadut asiat ottaa käsittelyyn pikaisesti ja ongelmitta.

Osana tätä opinnäytetyötä esiteltiin myös pienyrittäjän näkökulmasta tilaustyönä tehty ohjelmisto, soveltaen kirjallisuuden neuvoja. Projektin aikana tehdyt empiiriset havainnot tukevat näkökantaa projektinhallinnan menetelmän sekä taustatyön merkityksestä vastatessa asiakastarpeisiin pienyrittäjän näkökulmasta. On kuitenkin tärkeää huomata kyseisen projektin todisteen anekdoottisuus. Ohjelmiston tuottaminen kuitenkin on ollut onnistunutta sekä luontevaa kirjallisuuden oppien avulla, kun huomioitiin haasteet mitkä tuli esille kirjallisuutta tutkiessa.

Pienyrittäjän ohjelmistotuotannossa erittäin oleellista on oletettavasti tutustuminen asioihin mitkä saattavat projektin pilata. Pienyrittäjälle on varmaankin tärkeämpää keskittyä välttämään suurimmilta virheiltä (suunnittelun puute, kommunikoinnin puute, kokemattomat tiimit, perinteinen projektinhallinta ja projektipäällikön vastuu monessa asiassa (Rowe, 2015)) ja iteratiivisesti parantaa heille sopivaa toimintamallia sen sijaan että suoraan tavoittelisi täydellistä tuotantoprosessia. Asiakastarpeisiin vastaaminen onnistunee parhaiten taustatyön, kommunikaation (sekä tiimin sisäisen että asiakaskommunikoinnin), projektinhallinnan sekä pätevän tiimin avulla.

LÄHTEET

1. Ahmad, M., Markkula, J., Oivo, M. (2013). Kanban in software development: A systematic literature review. 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, [online] p.3. Available at: https://www.researchgate.net/publication/260739586_Kanban_in_Software_Development_A_Systematic_Literature_Review [Accessed 24 Nov. 2018].
2. Banijamali, A., Dawadi, R., Ahmad, M., Similä, J., Oivo, M., Liukkunen, K. (2017). Empirical Investigation of Scrumban in Global Software Development. In: Hammoudi S., Pires L., Selic B., Desfray P. (eds) Model-Driven Engineering and Software Development. MODELSWARD 2016. Communications in Computer and Information Science, vol 692. Springer, Cham. Available at: <https://www.researchgate.net/publication/319641301> [Accessed 7 Dec. 2018].
3. Beyer, C. (2018). The Node.js Ecosystem Is Chaotic and Insecure. [online] Medium. Available at: <https://medium.com/commitlog/the-internet-is-at-the-mercy-of-a-handful-of-people-73fac4bc5068> [Accessed 7 Dec. 2018].
4. Boehm, B. (2002). Get ready for agile methods, with care. Computer, [online] 35(1), pp.64-69. Available at: <https://www.researchgate.net/publication/2955570>.
5. Brezočnik, L., Majer, Č. (2016). Comparison of agile methods: Scrum, Kanban, and Scrumban. In: Proceedings of the 19th International Multiconference Information Society - IS 2016: Collaboration, software and services in information society, October 20, Ljubljana, Slovenia, 2016. p.3.
6. Nodejs.org. (2018). C++ Addons | Node.js v11.3.0 Documentation. [online] Available at: <https://nodejs.org/api/addons.html> [Accessed 7 Dec. 2018].
7. Caballero, E., Calvo-Manzano, J., San Feliu, T. (2011). Introducing Scrum in a Very Small Enterprise: A Productivity and Quality Analysis. Systems, Software and Service Process Improvement, [online] pp.215-224. Available at: https://www.researchgate.net/publication/221046001_Introducing_Scrum_in_a_Very_Small_Enterprise_A_Productivity_and_Quality_Analysis [Accessed 15 Nov. 2018].
8. Coleman, G., O'Connor, R. (2008). Investigating software process in practice: A grounded theory perspective. Journal of Systems and Software, [online] 81(5), pp.772-784. Available at: <https://doi.org/10.1016/j.jss.2007.07.027>.

9. Erickson, J., Lyytinen, K., Siau, K. (2005). Agile Modeling, Agile Software Development, and Extreme Programming. *Journal of Database Management*, [online] 16(4), pp.88-100. Available at: https://www.researchgate.net/publication/220373708_Agile_Modeling_Agile_Software_Development_and_Extreme_Programming_The_State_of_Research [Accessed 2 Dec. 2018].
10. Fair, J. (2012). Agile versus Waterfall: approach is right for my ERP project? Paper presented at PMI® Global Congress 2012—EMEA, Marsailles, France. Newtown Square, PA: Project Management Institute. Available at: <https://www.pmi.org/learning/library/agile-versus-waterfall-approach-erp-project-6300> [Accessed 7 Dec. 2018].
11. Haikala, I., Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt*. 12th ed. pp.21, 61-68.
12. Hinkelmann, F. (2018). Speed up Your Node.js App with Native Addons. [online] Medium. Available at: <https://medium.com/the-node-js-collection/speed-up-your-node-js-app-with-native-addons-5e76a06f4a40> [Accessed 7 Dec. 2018].
13. GitHub. (2018). I don't know what to say. · Issue #116 · dominictarr/event-stream. [online] Available at: <https://github.com/dominictarr/event-stream/issues/116> [Accessed 7 Dec. 2018].
14. Kniberg, H., Skarin, M., Poppendieck, M., Anderson, D. (2010). *Kanban and Scrum*. C4Media, pp.11-17.
15. Korkala, M. (2015). Customer communication in distributed agile software development. Espoo: Teknologian tutkimuskeskus VTT Oy, pp.50-52, 56-60.
16. Lee, S., Yong, H. (2013). Agile Software Development Framework in a Small Project Environment. *Journal of Information Processing Systems*, 9(1), pp.69-88.
17. MacCormack, A., Verganti, R., Iansiti, M. (2001). Developing Products on “Internet Time”: The Anatomy of a Flexible Development Process. *Management Science*, 47(1), pp.133-150.
18. Agilemanifesto.org. (2001). Manifesto for Agile Software Development. [online] Available at: <https://agilemanifesto.org/> [Accessed 1 Dec. 2018].
19. Paetsch, F., Eberlein, A., Maurer, F. (2003). Requirements Engineering and Agile Software Development. In: WET ICE 2003. Proceedings. Twelfth IEEE

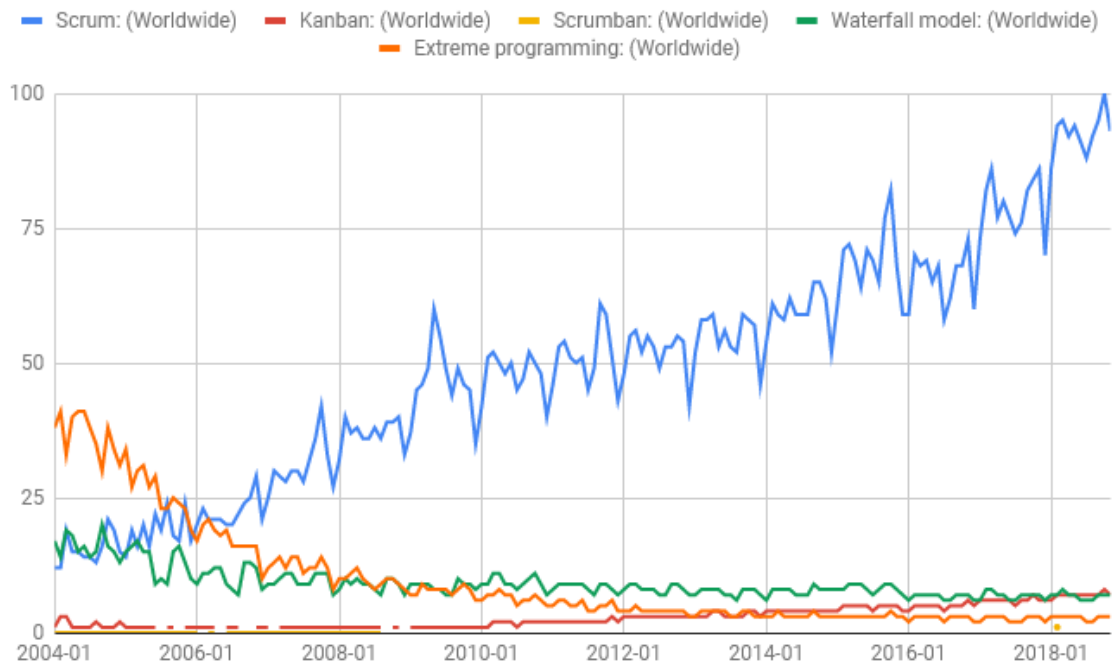
- International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 11 June, Linz, Austria 2003. [online] IEEE, p.1. Available at: <https://ieeexplore.ieee.org/abstract/document/1231428/> [Accessed 2 Dec. 2018].
20. Pino, F., García, F., Piattini, M. (2007). Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal*, [online] 16(2), pp.237-261. Available at: <https://www.researchgate.net/publication/220635980> [Accessed 3 Dec. 2018].
 21. Richardson, I., Von Wangenheim, C. (2007). Guest Editors' Introduction: Why are Small Software Organizations Different?. *IEEE Software*, [online] 24(1), pp.18-22. Available at: <https://www.researchgate.net/publication/3249286>.
 22. Rogers, S. (2016). What is Google Trends data — and what does it mean?. [online] Medium. Available at: <https://medium.com/google-news-lab/what-is-google-trends-data-and-what-does-it-mean-b48f07342ee8> [Accessed 7 Dec. 2018].
 23. Rowe, S. (2015). *Project management for small projects*. Tysons Corner, Virginia: Management Concepts Press.
 24. Rubin, K. (2012). *Essential Scrum*. Upper Saddle River, NJ: Addison-Wesley, pp.1-4, 13-16.
 25. Satpathyt, (2014), Overview of a Scrum project's flow [ONLINE]. Available at: https://commons.wikimedia.org/wiki/File:Scrum_Flow_for_one_Sprint.png [Accessed 7 December 2018].
 26. Shore, J., Warden, S. (2008). *The Art of agile development*. 1st ed. Beijing: O'Reilly, pp.121-125.
 27. The Standish Group International, Inc (1995). *The CHAOS report*. [online] Available at: https://www.researchgate.net/publication/263849222_The_Chaos_Report [Accessed 7 Dec. 2018].
 28. Turner, R., Ledwith, A., Kelly, J. (2009). Project management in small to medium-sized enterprises: A comparison between firms by size and industry. *International Journal of Managing Projects in Business*, [online] 2(2), pp.282-296. Available at: <https://www.researchgate.net/publication/235317151>.
 29. Turner, R., Ledwith, A., Kelly, J. (2012). Project management in small to medium-sized enterprises: Tailoring the practices to the size of company. *Management*

- Decision, [online] 50(5), pp.942-957. Available at:
<https://www.researchgate.net/publication/263558067>.
30. Vijayasarathy, L.E.O.R., Turk, D., 2008. Agile software development: A survey of early adopters. *Journal of Information Technology Management*, 19(2), pp.1-8.
 31. Vahaniitty, J., Rautiainen, K., Lassenius, C. (2010). Small software organizations need explicit project portfolio management. *IBM Journal of Research and Development*, [online] 54(2), pp.1:1-1:12. Available at:
<https://www.researchgate.net/publication/224127481>.
 32. van Waardenburg, G., van Vliet, H. (2013). When agile meets the enterprise. *Information and Software Technology*, [online] 55(12), pp.2154-2171. Available at:
<https://www.sciencedirect.com/science/article/abs/pii/S0950584913001584?via%3Dihub>.
 33. Planisware. (2018). What is a project portfolio? | PPM Glossary. [online] Available at: <https://www.planisware.com/glossary/project-portfolio> [Accessed 7 Dec. 2018].
 34. Pmi.org. (2018). What is Project Management?. [online] Available at: <https://www.pmi.org/about/learn-about-pmi/what-is-project-management> [Accessed 7 Dec. 2018].
 35. LeanKit. (2018). What is Scrumban?. [online] Available at: <https://leankit.com/learn/agile/what-is-scrumban/> [Accessed 7 Dec. 2018].

**LIITE 1. Lean johtamisfilosofian ja Kanbanin periaatteet. Ahmad et al., (2013),
käännetty lähteestä.**

Lean johtamisfilosofian seitsemän periaatetta	Kanbanin periaatteet
Eliminoidi hukkaa	Visualisoi työnkulku
Kunnioita ihmisiä	Rajoita keskeneräistä työtä
Lykkää sitoutumista	Mittaa ja hallitse sujuvuutta
Luo laadukasta	Tee prosessin menettelytavasta täsmällinen
Toimita nopeasti	Kehitä yhteistyötä
Luo tietämystä	
Optimoi kokonaisvaltaisesti	

LIITE 2.



LIITE 3.



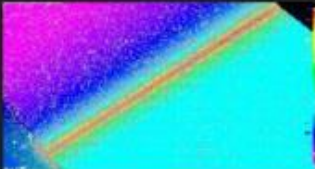
LIITE 4. Alpha versio luodusta ohjelmistosta

✗ Material selection


SELECT

Select a material to proceed

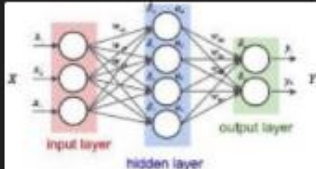
✗ Structural design



FEA ?
SELECT



Measured ?
SELECT



ANN ?
SELECT

STATIC DESIGN

FATIGUE DESIGN

⋮ Design of fabrication

CUTTING

COLD FORMING

MACHINING

WELDING

POST TREATMENTS

INSPECTION