Agile-menetelmien soveltaminen sulautettujen järjestelmien laitteistokehitykseen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietotekniikan Koulutusohjelma

Forssa, syksy 2016

Antti Teronen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Forssa
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät

Tekijä Antti Teronen **Vuosi** 2016

Työn nimi Agile-menetelmien soveltaminen sulautettujen järjestelmien laitteistokehitykseen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli tutustua ohjelmistokehityksessä käytettyihin työskentelymenetelmiin ja tutkia niiden sovellettavuutta sulautettujen järjestelmien laitteistokehitykseen.

Motivaationa työn aiheen valinnalle oli tekijän oman työuran aikana muotoutuneiden työskentelytapojen haastaminen ja tutustuminen menetelmiin, joilla elektroniikkasuunnittelua voisi tehdä helpommaksi ja nopeammaksi

Työn taustaksi tutustuttiin kolmeen ohjelmistokehityksen menetelmään, joissa sovelletaan jatkuvan ja lisäävän kehityksen periaatteita. Lisäksi tutustuttiin ketterän ohjelmistokehityksen julistukseen. Saatavilla olevasta kirjallisuudesta etsittiin myös jo dokumentoituja menetelmiä tai ajatuksia ohjelmistokehityksen ketterien menetelmien soveltamisesta elektroniikan suunniteluun. Lisämateriaaliksi haastateltiin tekijän työyhteisön elektroniikkasuunnittelijoita heidän hyviksi kokemiensa menetelmien ja käytäntöjen keräämiseksi opinnäytetyön taustatiedoiksi.

Opinnäytetyön johtopäätös on, että ohjelmistokehityksen ketteriä menetelmiä voidaan hyödyntää sulautettujen järjestelmien laitteistokehityksessä, mutta ei aivan sellaisenaan.

Ohjelmistokehityksen tapa jaksottaa kehitys sopivan pituisiin jaksoihin ja julkaista uusi versio jakson päätteeksi vaatii laitteistokehityksen yhteydessä erilaista toimintatapaa. Agile-menetelmien sovellettaessa laitteistokehitykseen on huomioitava laitteiston prototyyppien rakentamiseen liittyvät odotusajat ja kustannukset. Toisaalta iso osa ohjelmistokehityksen ketteristä menetelmistä voisi auttaa myös laitteistokehityksessä. Dokumentaation pitäminen kevyenä, vaatimusmäärittelyn täydentäminen joustavasti projektin aikana, tilaajan ja työryhmän yhteistyön parantamiseen tähtäävät menetelmät, testattavuuden ja testausautomaation kehittäminen sekä työryhmän itseohjautuvuus voivat kaikki parantaa projektin onnistumisen mahdollisuuksia.

Avainsanat Ketterä kehitys, laitteistokehitys, ohjelmistokehitys

Sivut 40 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Forssa
Degree Programme in Information Technology
Embedded systems

Author Antti Teronen **Year** 2016

Subject of Bachelor's thesis Applying Agile development methods to the hardware development of embedded systems

ABSTRACT

The purpose of this thesis project was to get familiarized with the working practices and methods used in software development, and to study their applicability to the development of hardware in embedded systems.

The motivation for selecting this topic was a desire to challenge the working methods and practices the author had adopted during his working career, and to find ways of making electronics design work easier and faster.

As background material for the research project three iterative and incremental development methods and the Agile development manifesto and principles were studied. Available literature and materials were examined to find examples and experience about applying Agile software development methods applications to the development of electronics. Additional material about good working practices and methods were gathered by interviewing the electronics and system design engineers in author's working community.

The conclusion from the research project was that it is possible to use Agile methods on the hardware development of embedded systems, but not directly as they are used in software development. The development method of sequencing it in to short time boxed iterations and releasing a new version at the end of the iteration requires a different approach in hardware development. When applying Agile methods to hardware development, one must take into account the lead time and cost of building prototypes.

On the other hand most of the other Agile methods used in software development could be of use in hardware development as well. Keeping the documentation light, supplementing requirements flexibly during the project, methods for improving collaboration between the customer and the development team, improving testability and test automation and self-guidance of the development team can all improve the chances of the project to be completed successfully.

Keywords Agile development, hardware development, software development

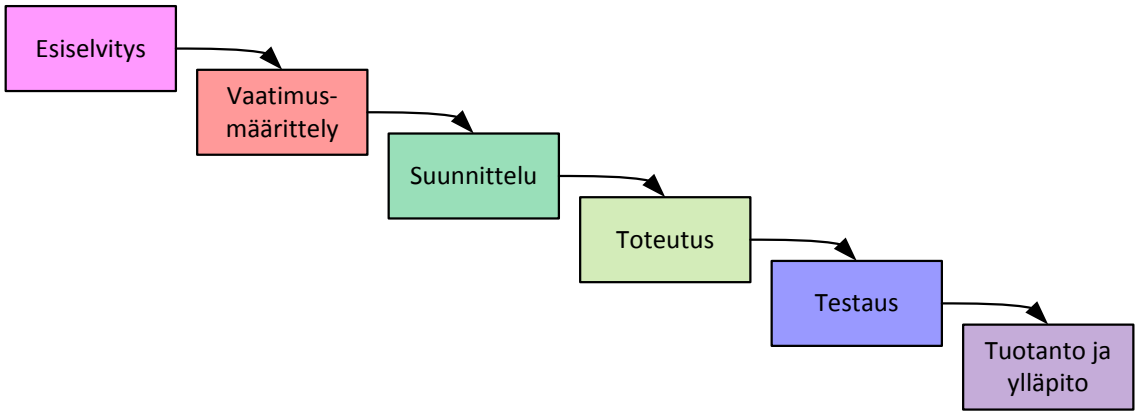
Pages 40 p.

SISÄLLYS

1 JOHDANTO

Valmistuin insinööriksi vuonna 1999 Helsingin teknillisestä oppilaitoksesta. Halusin päivittää tutkintoni AMK insinööriksi, jonka vuoksi aloitin opiskelut Hämeen ammattikorkeakoulussa vuonna 2012.

17 vuoden työurani aikana olen toiminut useammassa yrityksessä elektroniikan ja järjestelmien suunnittelutehtävissä. Jokaisessa yrityksessä on ollut omat tuotekehitysprosessinsa, mutta pääasiassa ne ovat mukailleet nk. vesiputousmallia. Vesiputousmalli (Kuva 1) yksinkertaistaa tuotekehityksen yleensä kuuteen vaiheeseen, joiden valmistumista seurataan ja vaiheesta toiseen siirtyminen päätetään vaihekatselmuksissa.



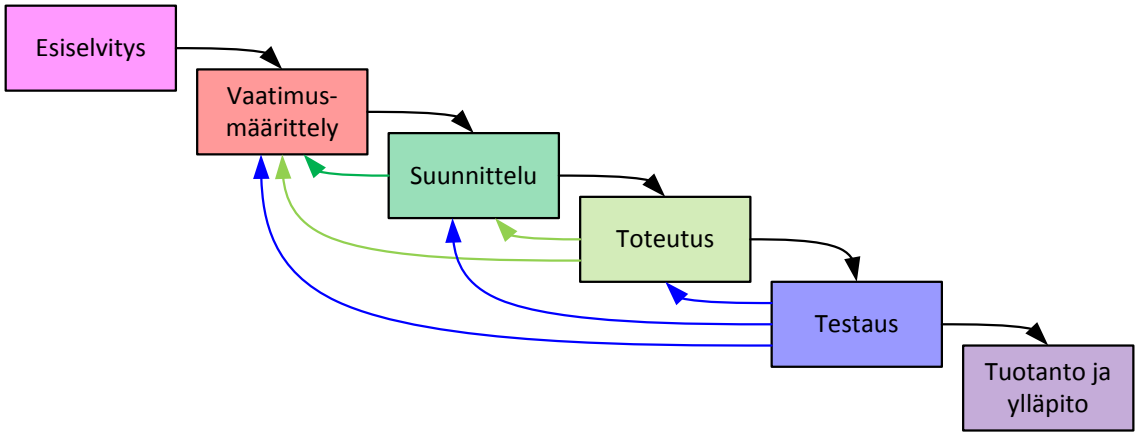
Kuva 1: **Vesiputousmalli**

Vesiputousmallin ongelma järjestelmien ja laitteistojen kehityksessä on se, että se soveltuu vain kaikkein yksinkertaisimpiin toteutuksiin, joissa on aidosti mahdollista määritellä vaatimukset etukäteen kattavasti eikä toteutuksessa ole yhtään riskiä.

Työurani aikana olen huomannut, että vesiputousmallia ei ole pystytty soveltamaan dokumentoidulla tavalla yhteenkään projektiin jossa olen ollut mukana. Eri tuotekehitysprosessien määrittelemät vaihekatselmukset ovat jääneet poikkeuksetta vajaiksi ja korjaaviin sekä seuraavan vaiheen aloittamiseksi vaadittaviin toimenpiteisiin on yleensä otettu jonkinlainen aikalisä. Aikataulun kirimiseksi kiinni tuotteesta on pudotettu pois ominaisuuksia tai tuotteen testauksen kattavuudesta on tingitty.

Vesiputousmallia ei siis ole pystytty noudattamaan tai se on aiheuttanut sen, että projektin sisällä tehdään erilaisia kompromisseja ja runsaastikin lisätyötä ilman kunnollista tietopohjaa jonkin teknisen tai toteutukseen liittyvän riskin pienentämiseen. Sama työ olisi voitu tehdä projektissa hieman eri vaiheessa, kun esim. prototyypin mittausten perusteella tiedot päätöksentekoon olisivat olleet saatavilla.

Käytännön vesiputousmallia voisikin ajatella lähinnä kahtena putouksena, joiden välillä on suvanto. Suvannossa on akanvirtoja, joissa vesi virtaa vastakkaiseen suuntaan (Kuva 2).



Kuva 2: **Mukailtu vesiputousmalli, jossa palataan korjaamaan tai muuttamaan edellisen vaiheen tuotoksia.**

Laitteistokehitysprojekteissa joissa olen ollut mukana, on aina ollut myös ohjelmistokehitystä. Ohjelmistokehityksen projektit ovat käyttäneet erilaisia menetelmiä ja nykypäivänä lähes poikkeuksetta käytössä on jokin ketterä menetelmä tai yhdistelmä eri menetelmien käytäntöjä.

Sivusta katsottuna eri ohjelmistokehityksen menetelmät ovat onnistuneet vaihtelevasti toimittamaan toimivaa ohjelmakoodia ajoissa, mutta aina kun ohjelmiston on toimittanut pienehkö tiivistä yhteistyötä tehnyt ryhmä, on onnistumisen mahdollisuus ollut hyvä. Tällaisten työryhmien työtä seuratessa ovat keskusteluissa vilahdelleet sanat ”*Agile*”, ”*sprint*”, ”*backlog*” ja ”*burndown*”, jotka kaikki liittyvät ohjelmistokehityksen toimintatapojen ympärillä käytävään vilkkaaseen mielipiteiden vaihtoon.

Ymmärtääkseni paremmin, mistä ketterien ohjelmistokehityksen menetelmien yhteydessä puhutaan, miten ne ovat kehittyneet ja miettiessäni miten laitteistokehityksen toimintatapoja voisi parantaa, heräsi kiinnostus tutustua jatkuvan ja lisäävän kehityksen menetelmiin sekä Agile-julistuksen periaatteisiin. Tämän takia opinnäytetyöni aiheeksi valikoitui Agile-menetelmien soveltaminen sulautettujen järjestelmien laitteistokehitykseen.

Seuraavissa luvuissa on esitelty taustatiedoiksi kolme jatkuvan ja lisäävän kehityksen menetelmää, Agile-julistuksen periaatteet sekä tällaisten menetelmien jo olemassa olevia sovelluksia laitteistokehityksessä.
Lisämateriaaliksi opinnäytetyötä varten haastattelin työnantajani GE Healthcare Finland Oy:n elektroniikkasuunnittelijoita heidän hyväksi havaitsemien suunnittelu- ja kehitysmenetelmien kartoittamiseksi.

Kerätyn materiaalin ja taustatietojen perusteella on listattu parannusehdotuksia laitteistosuunnittelun eri tehtäviin.

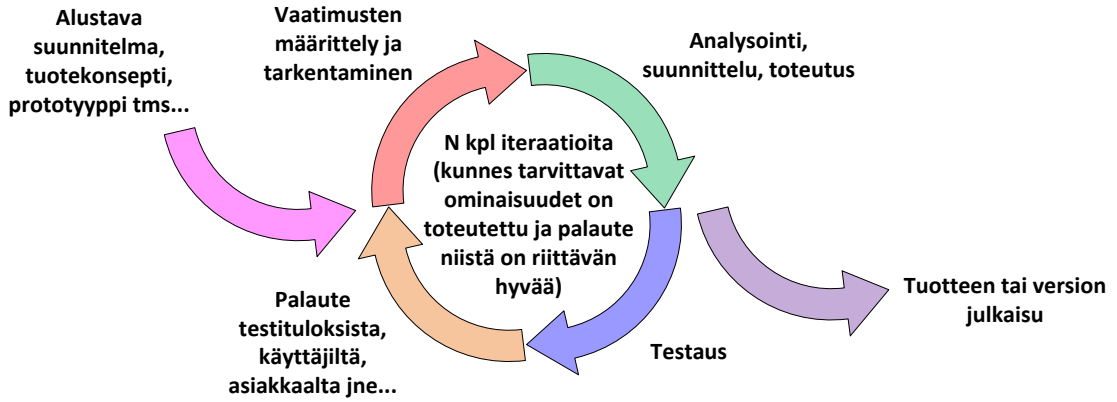
2 JATKUVAT JA LISÄÄVÄT KEHITYSMENETELMÄT

IID-menetelmissä (*Iterative and Incremental Development*) ohjelmisto, laite tai vaikka kokonainen järjestelmä kehitetään sarjana peräkkäisiä iteraatioita.

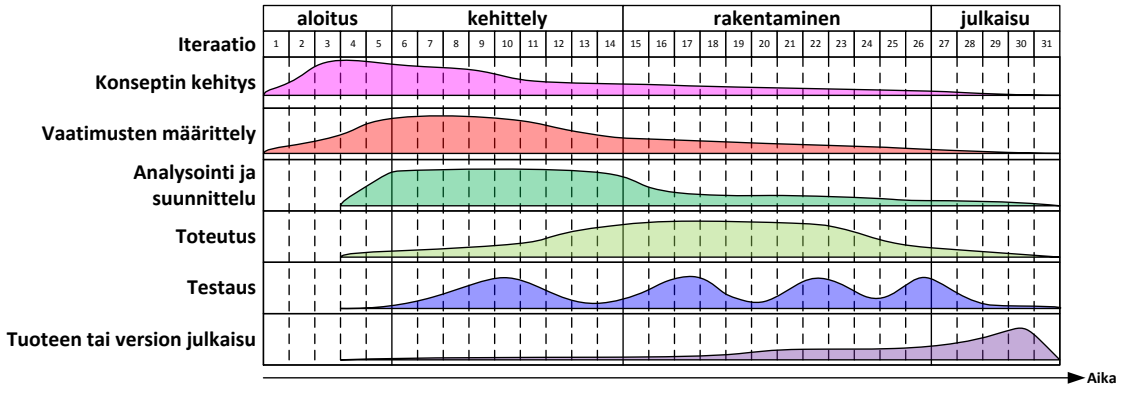
Jokaisen iteraation tavoitteena on päästä lähemmäs valmista toimitettavaa tuotetta tai lisätä jo toimivaan tuotteeseen ominaisuuksia tai korjauksia. IID voitaisiin suomentaa vaikka jatkuva ja lisäävä kehitysmenetelmä.

Jokaisen iteraation voidaan ajatella olevan oma pienoisprojektinsa, jonka kuluessa käydään läpi aktiviteetteja kuten vaatimusmäärittely, suunnittelu, toteutus ja testaus. IID-menetelmissä ei siis välttämättä pyritä esimerkiksi tekemään vaatimusmäärittelyä koko järjestelmälle etukäteen tai kirjoittamaan dokumentointia kerralla valmiiksi, kuten monien yritysten vesiputousmalliin perustuvissa tuotekehitysprosesseissa. Suurien kokonaisuuksien sijasta työ pyritään jakamaan niin, että osia tuotteesta tai järjestelmästä tehdään paloina, sekä niin että palaset ovat itsenäisiä ominaisuuksia joiden toteutus ja testaaminen erikseen ovat mahdollisia.

IID-menetelmiä havainnollistetaan usein kuvilla (Kuva 1Kuva 3; Kuva 4), joissa ilmenee menetelmien ominaispiirre, eli kaikkien tuotekehityksen normaalien osa-alueiden toistuminen jokaisen iteraation aikana.



Kuva 3. **IID-menetelmän kuvaus toistuvina iteraatioina, joista jokainen vie lähemmäs julkistettavaa tuotetta.**



Kuva 4. **UP prosessin kuvaus, joka havainnollistaa eri aktiviteettien määrää tuotekehityksen vaiheiden ja iteraatioiden aikana.**

Kuva 3 on mukaelma englanninkielisen Wikipedian IID-artikkelissa esitetystä kuvasta.

Kuva 4 on mukaelma englanninkielisen Wikipedian UP (*Unified Process*) ohjelmistokehitysprosessin artikkelissa esitetystä kuvasta.

Yhteistä useimmissa IID-menetelmissä on pitää jokainen iteraatio niin suppeana, että kehityksen parissa työskentelevät ihmiset pystyvät hahmottamaan selkeästi iteraation sisällön, sekä näkevät sen sisällön toimittamisen mahdollisena ennalta sovitun ajanjakson kuluessa (*time boxing*).

Suurimpana etuna IID-menetelmissä pidetään niiden mukautuvuutta muuttuviin vaatimuksiin. Jokaisen iteraation alussa määritellään mitä iteraation kuluessa halutaan saavuttaa. Tämä mahdollistaa myös uusien vaatimusten ottamisen kehityksen alle. Toinen etu IID-menetelmistä on kehitystyön kuluessa saatu säännöllinen palaute. Iteraation lopussa tehtävä testaus ja sen tulokset antavat mahdollisuuden siirtää korjaukset ja muutosehdotukset heti seuraavaan iteraatioon. Kun jokaisen iteraation tavoitteena on toimittaa toimiva kokonaisuus, on seuraavan iteraation lähtökohta selkeä ja uusien vaatimusten ottaminen mukaan on helpompaa.

IID-menetelmiä on käytetty ohjelmistokehityksessä useita vuosikymmeniä. Erilaisia menetelmiä on kehittynyt ja dokumentoitu useita ja näiden historiasta on kirjoittanut esim. Craig Larman hyviä yhteenvetoja (Larman 2012, 63-107; Basili & Larman 2003).

IID-menetelmät soveltuvat erityisen hyvin ohjelmistojen kehitykseen. Lähes kaikki dokumentoidut IID-menetelmät onkin kehitetty ohjelmistojen kehityksen työkaluiksi. Syy tähän on hyvin selkeä – ohjelmistokehityksessä iteraatiot voidaan pitää lyhyinä, koska tuote on ns. aineeton ja usein jopa laitteistoriippumaton. Tällöin peräkkäiset iteraatiot voidaan pitää aina samanpituisina, sisällöltään samankaltaisina, toimituslaajuudeltaan selkeästi rajattuna ja sopeutettuna käytössä oleviin resursseihin, sekä iteraation lopputuotteena voi aina olla toimiva ja testattu versio ohjelmistosta.

Laitteistokehityksessä IID-menetelmien soveltaminen on huomattavasti hankalampaa. Haasteita ovat lähinnä laitteistokehitykseen prototyyppien rakentamisen kustannukset ja siihen kuluva aika. Valmistamiseen kuluvan ajan ja kustannuspaineiden takia laitteiston elektroniikan ja mekaniikan kehityksessä ajaudutaan tekemään suurempia kokonaisuuksia kerralla yhteen prototyyppikierrokseen. Iteraatioiden keston pidentyessä ja laajuuden kasvaessa niiden hallintaan tarvitaan enemmän resursseja.

2.1 Ketterä kehitys

Nykyään puhuttaessa ketterästä tai Agile-kehityksestä, tarkoitetaan usein jotain IID-menetelmää sekä vuonna 2001 tehdyn Agile-julistuksen (*Agile manifesto*) periaatteiden soveltamista.

Agile-julistus on 17 ohjelmistokehittäjän vuonna 2001 tekemä lista arvoista ja periaatteista, joiden avulla ohjelmistokehityksen työskentelytapoja voi parantaa. Agile-julistus ja myös sen suomennettu versio löytyy www-sivulta http://agilemanifesto.org/.

Agile-julistuksen sisältö oli 4 paria arvoja, joista toista puolta julistuksen allekirjoittajat kuitenkin asettavat etusijalle. Agile-julistuksen sisältö oli:

> *Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä*
> *itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:*
>
> – *Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja*
> – *Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota*
> – *Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja*
> – *Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa*
>
> *Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän.*

(Beck & Beedle & van Bennekum & Cockburn & Cunningham & Fowler & Grenning & Highsmith & Hunt & Jeffries & Kern & Marick & Martin & Mellor 6 Schwaber & Sutherland & Thomas. 2001.)

Agile-julistuksen kanssa listattiin 12 periaatetta. Periaatteet julistuksessa olivat:

> *Noudatamme seuraavia periaatteita:*
> – *Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.*
> – *Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.*
> – *Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.*
> – *Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.*
> – *Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.*
> – *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*

- *Toimiva ohjelmisto on edistymisen ensisijainen mittari.*
- *Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.*
- *Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.*
- *Yksinkertaisuus – tekemättä jätettävän työn maksimointi – on oleellista.*
- *Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.*
- *Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.*

(Beck jne. 2001.)

2.2 Esimerkkejä iteratiivisista ja lisäävistä menetelmistä ohjelmistokehityksessä

Seuraavissa luvuissa on esitetty muutaman ohjelmistokehityksessä käytetyn IID-menetelmän pääperiaatteita.

2.2.1 UP – Unified Process

UP (*Unified Process*) on suosittu IID-menetelmä, mistä on dokumentoitu useita erilaisia muunnelmia.

Tunnetuin ja parhaiten dokumentoitu muunnelma lienee RUP™ (*Rational Unified Process*), joka on IBM:n omistama tuotemerkki. Muita muunnelmia ovat esimerkiksi OpenUP ja Agile UP. UP-menetelmää kuvataan tuotekehityksen toimintatavan rungoksi, joka tulisi muokata sitä käyttävän organisaation omiin tarpeisiin.

Ensimmäinen teos, joka käsitteli UP-menetelmää, oli 1999 julkaistu *The Unified Software Development Process*. Sen kirjoittajina olivat Ivar Jacobson, Grady Booch ja James Rumbaugh.

UP jakaa kehitysprojektin neljään vaiheeseen. Projektin vaiheet ovat:
- aloitus (*inception*)
- kehittely (*elaboration*)
- rakentaminen (*construction*)
- julkaisu (*transition*).

Menetelmä korostaa, että edellä mainitut neljä vaihetta eivät vastaa nk. vesiputousmallin vaatimusmäärittely-, suunnittelu-, ohjelmointi- sekä testausvaiheita (Larman 2012, 180-181, 194). Kehittely, rakentaminen ja julkaisu jaetaan IID-menetelmien tapaan sarjaksi ennalta määrätyn pituisia jaksoja. Jokaisen jakson tuloksena pitäisi olla edelliseen jaksoon verrattuna järjestelmä tai ohjelmisto, jonka ominaisuuksia on parannettu tai johon on lisätty uusia toiminnallisuuksia.

Prosessia kuvataan arkkitehtuuriin ja riskeihin keskittyväksi. Tämä näkyy siinä, että järjestelmän rakentamisvaiheeseen ei siirrytä ennen kuin arkki-

tehtuurin toimivuus on testattu käytännössä ja tunnistetut tekniset riskit ovat hallinnassa.

Kuva 4 sivulla 3 havainnollistaa resurssien käyttöä UP-menetelmän mukaisessa projektissa. Vaatimusmäärittelyä esimerkiksi voidaan tarkentaa projektin aikana sitä mukaan kun tekemisen edistyessä opitaan uusia asioita, tai asiakas haluaa projektiin muutoksia.

Unified Process menetelmän käytännöt ja ohjeet korostavat seuraavia asioita:

- Kehitys tulisi tapahtua lyhyissä ennalta määrätyn pituisissa jaksoissa (*timeboxed iterations*).
- Kehityksen tulisi keskittyä alussa asiakkaalle tärkeimpiin ominaisuuksiin (*customer high value elements*) ja järjestelmän toteutuksen kannalta suurimpiin riskeihin, sekä hyödyntäen olemassa olevien komponenttien uudelleen käyttöä.
- Kaiken tekemisen pitäisi keskittyä lisäarvon tuottamiseen asiakkaalle.
- Muutokset tulisi huomioida projektin alussa.
- Työskentelyn tulisi tapahtua yhtenä työryhmänä.

(Larman 2012, 173.)

UP-menetelmä ottaa kantaa useisiin ohjelmistoprojektin toimintatapoihin. RUP esimerkiksi korostaa, että suurin osa dokumentaatiosta tehdään UML (*Unified Modelling Language*) malleina kirjoitettujen vaatimusmäärittelyjen tai kuvausten sijasta.

Unified Process menetelmästä löytyy erilaisia valmiiksi kehitettyjä työkaluja. Esimerkiksi IBM julkaisee työkalua nimeltä Rational Method Composer, jonka avulla organisaatio voi luoda omaan tuotekehitysprosessiinsa sopivan ohjeistuksen RUP:in käyttöön, sekä hyödyntää valmiita dokumenttimalleja.

UP prosessista on julkaistu myös useita oppaita. Näistä esimerkiksi IBM:n julkaisema *Rational Unified Process – Best Practices for Software Development Teams* kuvaa selkeästi menetelmän vaiheet, toimintatavat ja työskentelyä tukevat dokumentit. Dokumentti on saatavilla IBM:n verkkokirjastossa http://www.ibm.com/developerworks/rational/library/ .

2.2.2 Scrum

Scrum (lyhennys englannin kielen sanasta ***scrummage***) tarkoittaa alun perin rugbyn pelin aloitustilannetta rikkomuksen jälkeen tai kun palloa ei voida enää pelata. Scrum tilanteessa pelaajat menevät tiiviiseen muodostelmaan päät alhaalla ja tavoittelevat palloa pelin uudelleenaloitustilanteessa.

Tuotekehityksen yhteydessä scrum nimitystä käyttivät ensimmäisen kerran japanilaiset Hirotaka Takeuchi ja Ikujiro Nonaka vuonna 1986 ilmestyneessä artikkelissaan nimeltä *New New Product Development Game*. Artikkelissa kuvataan tuotekehityksen kokonaisvaltaista ja iteratiivista mene-

telmää, joka on tunnistettu japanilaisten ja USA:laisten yritysten tuotteiden kehityksen menetelmien tutkimuksesta.

Artikkelissaan Takeuchi ja Nonaka ovat tunnistaneet kuusi keskeistä teemaa, jotka ovat olleet keskeisiä onnistuneen tuotekehityksen kannalta. Nämä ovat:

- Organisaation epävakaus (työryhmälle annetut erittäin haastavat tavoitteet mutta toisaalta vapaus valita toteutustavat itse) (*built-in instability*)
- Itseorganisoivat projektityöryhmät (*self-organizing project teams*)
- Limitetyt tuotekehitysvaiheet (*overlapping development phases*)
- Yksilöiden, työryhmien ja osaamisalueiden oppiminen toisiltaan (*multilearning*)
- Työryhmien hienovarainen ohjaus (*subtle control*)
- Osaamisen siirto työnkierron avulla (*organizational transfer of learning*)

Artikkelissa kirjoittajat kuvaavat menetelmää kokonaisvaltaiseksi, tai ”rugby menetelmäksi”, jossa joukkue pyrkii etenemään kentällä yhtenä yksikkönä syötellen palloa toisilleen (Nonaka & Takeuchi, 1986).

Vuonna 1995 Jeff Sutherland ja Ken Schwaber esittelivät olio-ohjelmointiin liittyvän konferenssipaperin nimeltä Scrum methodology. Menetelmän pohjana oli 1990 luvun alussa Ken Schwaberin yrityksessään (Advanced Development Methods) kehittelemä menetelmä, sekä samaan aikaan Eastel Corporationissa samantyyppinen menetelmä, jota kehittivät Jeff Sutherland, John Scumniotales ja Jeff McKenna. Schwaber ja Sutherland kehittivät menetelmää nykymuotoonsa, joka tunnetaan yleisesti nimellä Scrum.

Nykyään Scrum-menetelmän periaatteet ovat levinneet laajaan käyttöön ja niitä on myös kaupallistettu. Esimerkiksi Scrum.org ja Scrumalliance.org sekä useat muut tahot tarjoavat koulutusta ja sertifiointeja Scrum-menetelmiin liittyen.

Menetelmän kehittäjät kuvaavat Scrum-menetelmä kokonaisvaltaiseksi menetelmäksi, joka on kehitetty empiirisen prosessinohjausmallin teorian pohjalle. Empiirinen malli korostaa sitä, että tietotaito kasvaa kokemuksesta ja päätökset tehdään olemassa olevan tiedon perusteella. Scrum-menetelmä käyttää jatkuvaa ja lisäävää lähestymistapaa tuottavuuden ja riskienhallinnan optimoimiseen. Empiirisen prosessinohjauksen kolme pääperiaatetta ovat läpinäkyvyys, tarkastukset ja mukautuvaisuus (Schwaber & Sutherland, 2016, 3).

Scrum-menetelmä määrittelee kolme roolia, joiden kautta korostuu kehitystyöryhmän itsenäisyys sekä asiakkaan tai loppukäyttäjän jatkuvan palautteen saamisen tärkeys. Nämä kolme roolia ovat:

Tuoteomistaja (*Product Owner*)
Tuoteomistajan tärkein tehtävä on varmistaa, että kehitystyöryhmän tekemä työ tuo lisäarvoa asiakkaalle ja loppukäyttäjälle. Tuoteomistaja vastaa

tuotteen tehtävälistasta (*Product Backlog*) (Schwaber & Sutherland, 2016, 4). Tuoteomistaja toimii siis tuotteen tilaajan edustajana ja vastaa tuotteen vaatimuksista.

Scrum-menetelmässä tuotteen vaatimustenhallinta pyritään hoitamaan tuoteomistajan kautta niin, että tuoteomistaja lisää tuotteen tehtävälistaan käyttäjäläheisiä vaatimuksia (*user story*) sekä antaa niille prioriteetit.

Tuoteomistajan ei pitäisi ottaa kantaa tuotteen tekniseen toteutukseen, koska Scrum-menetelmässä kehitystyöryhmän halutaan toimivan itsenäisesti ja löytävän parhaan ratkaisun ongelmaan.

Tuoteomistajan työajasta suurin osa kuluu yhteistyöhön eri sidosryhmien kanssa keräten palautetta, jonka avulla kehitystyöryhmä pystyy tekemään mahdollisimman sopivan tuotteen asiakkaalle.

Kehitystyöryhmä (*Development Team*)
Kehitystyöryhmä koostuu 3-9 henkilöstä, jotka muodostavat itseohjautuvan työryhmän. Kehitystyöryhmä päättää itse miten tuotteen tehtävälista toteutetaan valmiiksi tuotteeksi asti. Tavoitteena on koota työryhmä niin, että sen sisältä löytyy kaikki tarvittava osaaminen kehitystyöryhmälle annetun tuotteen tai osakokonaisuuden toteuttamiseen (Schwaber & Sutherland, 2016, 6).

Kehitystyöryhmän tehtäviä ovat esimerkiksi ongelman analysointi, ratkaisujen kehittäminen, suunnittelu, toteutus, testaus ja dokumentointi.

Scrum-menetelmä korostaa kehitystyöryhmän itseohjautuvuutta ja sisäistä tehtävänjakoa kulloiseenkin tehtävään sopivaksi.

Työryhmän sisällä voi olla eri tehtäviin erikoistuneita asiantuntijoita, mutta koko työryhmä on vastuussa tehtävälistan mukaisen tuotteen tai osakokonaisuuden toimittamisesta.

Scrum master
Scrum masterin tehtävänä on tehdä kehitystyöryhmän toiminta helpoksi sekä ratkoa ongelmia, jotka estävät tai hidastavat kehitystyöryhmän tehtävien suorittamista. Scrum master myös huolehtii tuotekehitysmenetelmän periaatteiden noudattamisesta ja käytännöistä.

Scrum masterin rooli ei ole työryhmän vetäjän tai projektipäällikön rooli, vaan toimiminen kehitystyöryhmän puskurina ulkoisille häiriötekijöille, jotka voisivat hidastaa tai häiritä kehitystyötä.

Scrum master avustaa myös tuoteomistajaa tuotteen tehtävälistan ylläpidossa niin, että kehitystyöryhmä ymmärtää tehtävälistaan kirjatut käyttäjävaatimukset.

Scrum-menetelmässä kehitystyö jaetaan maksimissaan 4 viikon pituisiin jaksoihin, joita kutsutaan nimellä pyrähdys (*Sprint*).

Scrum-menetelmässä jokainen iteraatio koostuu pyrähdyksen suunnittelusta (*Sprint Planning*), päivittäisistä työryhmäkokouksista (*Daily Scrum*), kehitystyöstä, pyrähdyksen katselmoinnista (*Sprint Review*) sekä tarkastelusta (*Sprint Retrospective*).

Pyrähdyksen suunnittelussa iteraatioon valitaan tuotteen tehtävälistalta (*Product backlog*) sopiva määrä käyttäjävaatimuksia, joille kehitystyöryhmä pystyy tekemään toteutuksen. Vaatimukset pilkotaan tehtäviksi ja muodostetaan pyrähdykselle tehtävälista (*Sprint backlog*).

Scrum-menetelmä korostaa kehitystyöryhmän itseohjautuvuutta ja esimerkiksi pyrähdyksen tehtävälistan ja tehtävien jaon eri kehittäjille työryhmä päättää itse.

Iteraation aikana kehitystyöryhmä pitää työn ohessa lyhyen tilannekatsauksen (*Daily Scrum*), jossa jokainen työryhmän jäsen kertoo edellisenä päivänä valmistuneen työn ja miten aikoo jatkaa siitä. Tilannekatsauksissa tarkastellaan myös asioita jotka estävät työn etenemistä ja jotka siirtyvät Scrum masterin huomion alle.

Iteraation lopussa kehitystyöryhmä järjestää sidosryhmien kanssa katselmoinnin (*Sprint review*), jonka osana on yleensä demonstraatio iteraation aikana toteutetuista toiminnallisuuksista. Jokaisen iteraation lopussa kehitystyöryhmä järjestää myös tarkastelun (*Sprint retrospective)*, jossa pohditaan miten työryhmän toimintaa voi kehittää.

Pyrähdyksen muodollisuuksien, eli päivittäisten tilannekatsauksien, katselmoinnin ja toiminnan tarkastelun, tarkoitus on parantaa kommunikaatiota, läpinäkyvyyttä sekä oppimista.

Scrum-menetelmän periaatteet ja toimintatavat on kuvattu Ken Scwaberin ja Jeff Sutherlandin ylläpitämässä oppaassa nimeltä The Scrum Guide™ – The Definitive Guide to Scrum: The Rules of the Game. Opasta päivitetään säännöllisesti ja se on käännetty usealle kielelle.

2.2.3 FDD – Feature Driven Developent

FDD-menetelmän (*Feature Driven Development*), eli toiminnallisuuskeskeisen kehitysmenetelmän, kehitti alun perin Jeff De Luca vuonna 1997 suuren pankkisovelluksen kehitystä varten.

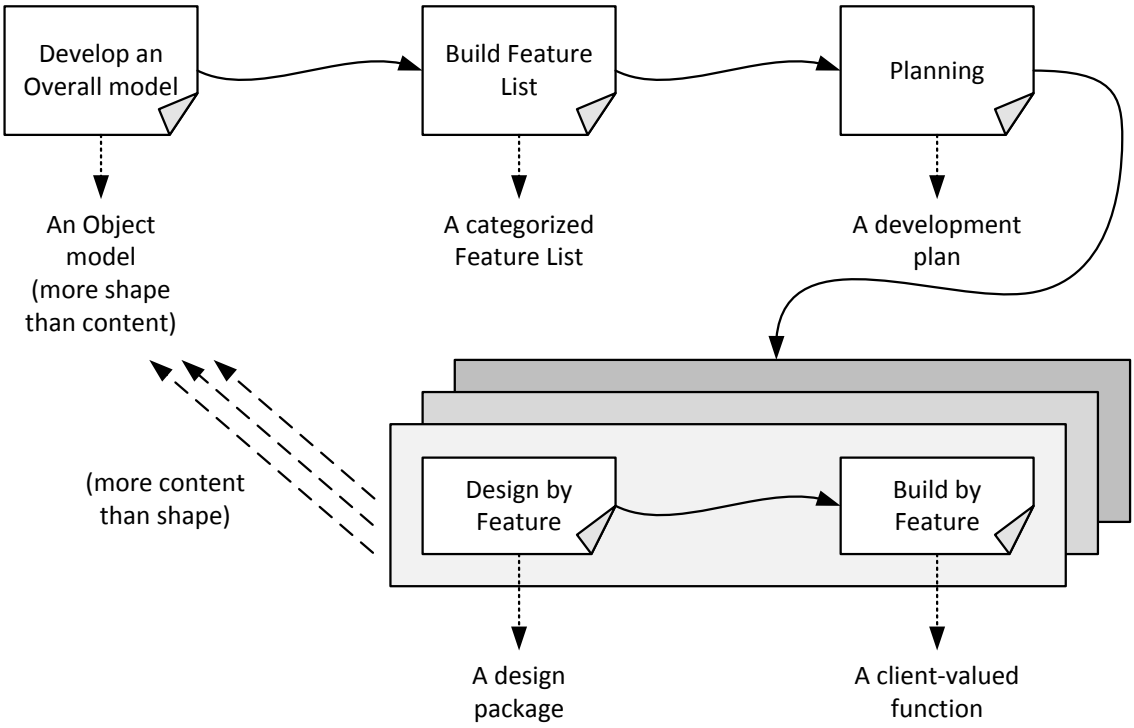
Alkuperäinen prosessi sisälsi paljon vaikutteita Peter Coadin ideoista ja esiteltiin vuonna 1999 teoksessa nimeltä Java Modeling In Color With UML, jonka kirjoittajina olivat Peter Coad, Eric Lefebvre ja Jeff De Luca.

FDD-menetelmää jatkokehittivät Stephen R. Palmer ja John M. Felsing, jotka julkaisivat 2002 teoksen nimeltä A Practical Guide to Feature-Driven Development. Tässä teoksessa FDD-menetelmä on irrotettu Java mallinnuksesta ja esitetty yleiskäyttöisempänä kehitysprosessimallina (Juhola, 2010, 14; Wikipedia, 2016).

FDD on menetelmä, joka rakentuu viiden perusaktiviteetin ympärille. Nämä aktiviteetit ovat:

1. Ylätason järjestelmämallin kehitys
2. Järjestelmän toiminnallisuuksien listaaminen
3. Kehityssuunnitelman teko toiminnallisuuksien perusteella
4. Suunnittelu ja toteutus toiminnallisuus kerrallaan
5. Järjestelmän testaus ja toimitus toiminnallisuus kerrallaan

Kuva 5 esittää FDD-menetelmän aktiviteetit kaaviona kuten menetelmän kehittäjä Jeff De Luca ne on esittänyt omassa esitysmateriaalissaan. Jokaiselle aktiviteetille on kehitetty tarkempia malleja, joita on esitetty esimerkiksi http://www.featuredrivendevelopment.com/ verkkosivulla.



Kuva 5. **FDD-prosessin yleiskuvaus (De Luca, 2006, 22)**

FDD-menetelmässä kaksi ensimmäistä aktiviteettia muodostavat projektin esiselvitysvaiheen, jossa pyritään määrittelemään ylätasolla järjestelmän arkkitehtuuri ja ominaisuudet.

Menetelmässä korostetaan hyvin paljon ohjelmiston mallinnuksen menetelmiä (oliomallit, sekvenssikaaviot, luokkakaaviot, nk. värimallit, jne.).

Kolmannessa aktiviteetissa luodaan suunnitelma, joka voi pitää sisällään esimerkiksi eri toiminnallisuuksien toteutusjärjestyksen.

Kahdessa viimeisessä aktiviteetissa korostuu IID-menetelmien tapa pyrkiä tekemään lopputuote pieninä kokonaisuuksina. FDD-menetelmässä suositellaan 2 viikon tai sitä lyhyempiä iteraatioita. Toteutettavat toiminnallisuudet tulisi rajata ja pilkkoa niin, että työryhmä pystyy tekemään ne valmiiksi iteraation aikana.

Jokaisen toiminnallisuuden toteutuksen yhteydessä havaitut puutteet ylätason malleissa voidaan korjata sekä täydentää ja ottaa huomioon seuraavissa iteraatioissa.

FDD-menetelmä määrittelee rooleja ja vastuita työryhmän jäsenille eri aktiviteettien aikana.
Työryhmän roolit ovat eri vaiheissa määritelty seuraavasti:

1. Ylätason järjestelmämallin kehitys
Projektipäällikkö (*Project Manager*) perustaa mallinnustyöryhmän, johon kuuluvat ***pääarkkitehti*** (*Chief Architect*), ***osakokonaisuuksien asiantuntijat*** (*Domain Experts*) ja ***johtavat ohjelmoitsijat*** (*Chief Programmers*). Mallinnustyöryhmä kehittää järjestelmän ylätason mallin sekä eri toiminnallisuuksien malleja. Mallinnustyön tuloksena olevat mallit katselmoidaan ***asiakkaan*** (*Business*) kanssa.

Järjestelmämallin kehityksen tuloksena saadaan ylätason ja toiminnallisuuksien kannalta tärkeiden kokonaisuuksien karkeat mallit ja riippuvuudet.

2. Järjestelmän toiminnallisuuksien listaaminen
Projektipäällikkö ja ***kehityspäällikkö*** (*Development Manager*) muodostavat ***johtavista ohjelmoitsijoista*** työryhmän toiminnallisuuksien listaamista varten. Toiminnallisuuksien listaamiseen käytetään FDD-menetelmässä erityistä muotoa ja toiminnallisuuksien pitäisi olla asiakkaalle lisäarvoa tuottavia.

Toiminnallisuuksien kirjoitusmuoto FDD-menetelmässä on määritelty seuraavasti:

<toiminto (action)> <tulos (result)> <kohde (object)>

Toiminnallisuus voi olla esimerkiksi ”Laske varastossa olevien tuotteiden varastoarvo raporttiin”. Toiminnallisuudet tulisi olla niin pieniä, että niiden kehitykseen ja testaamiseen riittää 2 viikon iteraatio. Toiminnallisuuslistan voi tarvittaessa katselmoida mallinnustyöhön osallistuneet ***osakokonaisuuksien asiantuntijat***.

3. Kehityssuunnitelman teko toiminnallisuuksien perusteella
Kehityssuunnitelman tekoa varten ***projektipäällikkö*** muodostaa työryhmän, jossa on jäseninä esimerkiksi ***kehityspäällikkö*** ja ***johtavat ohjelmoitsijat***.

Kehityssuunnitelmasta pitäisi selvitä eri toiminnallisuuksien toteutusjärjestys. Toteutusjärjestykseen vaikuttaa esimerkiksi toiminnallisuuksien riippuvuudet toisistaan ja niiden laajuus, sekä kehitysryhmien kuormitus eri toiminnallisuuksia kehitettäessä. Myös eri toiminnallisuuksien kehittämiseen liittyvät riskit otetaan huomioon ja suurimman riskin tehtävät pyritään tekemään ensin.

Suunnitelmasta tulisi selvitä myös toteutusaikataulu ja eri ***toiminnallisuuksien asiakkuusomistajat*** (*Business activity owner*). Omistajina toimivat johtavat ohjelmoitsijat.

Suunnitelmaan listataan myös yksittäisten ohjelmistokomponenttien omistajat, joina toimivat ***ohjelmistokehittäjät*** (*Developers*)

4. Suunnittelu ja toteutus toiminnallisuus kerrallaan

Toiminnallisuuden suunnittelu ja toteutus alkaa FDD-menetelmässä sillä, että toiminnallisuudesta vastaava ***johtava ohjelmoitsija*** listaa ohjelmistokomponentit, joita toiminnallisuuden toteutukseen tarvitaan. Johtava ohjelmoitsija ja tarvittavat ***kehittäjät*** muodostavat ***toiminnallisuuden kehitysryhmän*** (*feature team*).

Osana suunnittelun aloitusta johtava ohjelmoitsija myös luo uuden suunnittelupaketin (*design package*) kerätyistä tiedoista ja työryhmän kokoonpanosta, joka lisätään projektiin (*work package*).

Iteraation aluksi ***osakokonaisuuden asiantuntija*** esittelee toiminnallisuuden ja sen vaatimukset työryhmälle.

Varsinainen suunnittelu ja toteutus aloitetaan luomalla toiminnallisuudesta sekvenssikaaviot ja komponenttimallit, sekä päivittämällä koko järjestelmän mallia toteutettavan toiminnallisuuden osalta. Mallinnuksen jälkeen varsinainen ohjelmistokoodi voidaan kirjoittaa ja katselmoida sekä tehdä sen perusteella toiminnallisuudelle ohjelmointirajapintamäärittely (*Application Programming Interface (API) specification*).

Toiminnallisuuden suunnittelu- ja toteutusvaiheen tuloksena pitäisi olla kyseisen toiminnallisuuden mallit ja määrittelyt koko järjestelmän dokumentaatiota varten, sekä ohjelmistokoodi toiminnallisuuden integroimiseksi järjestelmään ja testausta varten.

5. Järjestelmän testaus ja toimitus toiminnallisuus kerrallaan

Toiminnallisuuden kehitysryhmä tekee uuteen toiminnallisuuteen liittyvälle ohjelmakoodille yksikkötestauksen sekä koodikatselmoinnin. Yksikkötestattu ja katselmoitu koodi integroidaan järjestelmään.

Lopputuloksena pitäisi olla toimiva järjestelmän toiminnallisuus, joka tuottaa lisäarvoa asiakkaalle.

FDD-prosessin tarkempi kuvaus, työmenetelmät ja roolit löytyvät Feature Driven Development processes kuvauksesta Nebulon USA LLC yrityksen WWW-sivulta http://www.nebulon.com/articles/fdd/latestprocesses.html. Myös esimerkiksi Tomi Juholan kirjaan (Juhola, 2010, 14-18) on koottu tarkka yhteenveto FDD-menetelmästä.

3 ELEKTRONIIKAN TUOTEKEHITYSMENETELMIÄ

Elektroniikan laitteistokehitys on tyypillisesti hyvin suoraviivaista ja se voidaan jakaa rajattuihin vaiheisiin vesiputousmallia mukaillen.

Prosessin vaiheet voivat olla esimerkiksi:

1. Tuoteidea
2. Esiselvitys
3. Arkkitehtuuri- ja järjestelmäkehitys
4. Vaatimusmäärittely
5. Suunnittelu, toteutus ja testaus
6. Tuotantoon siirto
7. Ylläpito

Tuoteidean esittely, esiselvitys sekä arkkitehtuuri- ja järjestelmäkehitys ovat valmistelevia vaiheita, joilla pyritän selvittämään tuotteeseen tarvittavat ominaisuudet ja toiminnallisuudet. Kolmen ensimmäisen vaiheen tuotoksena voi olla jo toimiva prototyyppi, jolla voidaan esitellä tärkeimpiä ominaisuuksia ja toiminnallisuuksia.

Neljäntenä vaiheena prosessissa voi olla vaatimusmäärittelyn tekeminen. Laitteistokehityksessä vaatimusmäärittelyn tekeminen on yleensä kohtalaisen suoraviivaista. Laitteiston määrittelee usein lopputuotteelle tai komponentille asetetut selkeät tavoitteet kuten:

- käyttöjännitteet ja tehonkulutus
- valmiusaika akkukäyttöiselle laitteelle
- laitteen tai piirikortin muoto, paino ja koko
- käyttöliittymän toteutustapa
- sähköiset liitynnät
- aikaisemman tuotteen komponenttivalinnat
- tuoteperheen sisällä uudelleenkäytettävät lohkot.

Puhtaasta vesiputousmallista poiketen laitteistokehityksen prosessin vaiheet limittyvät usein toistensa kanssa, tai niissä on piirteitä IID-menetelmistä.

Erityisesti vaatimusmäärittelyä tarkennetaan projektin kuluessa prototyyppien ja kehitysversioiden testauksesta saadun uuden tiedon perusteella.

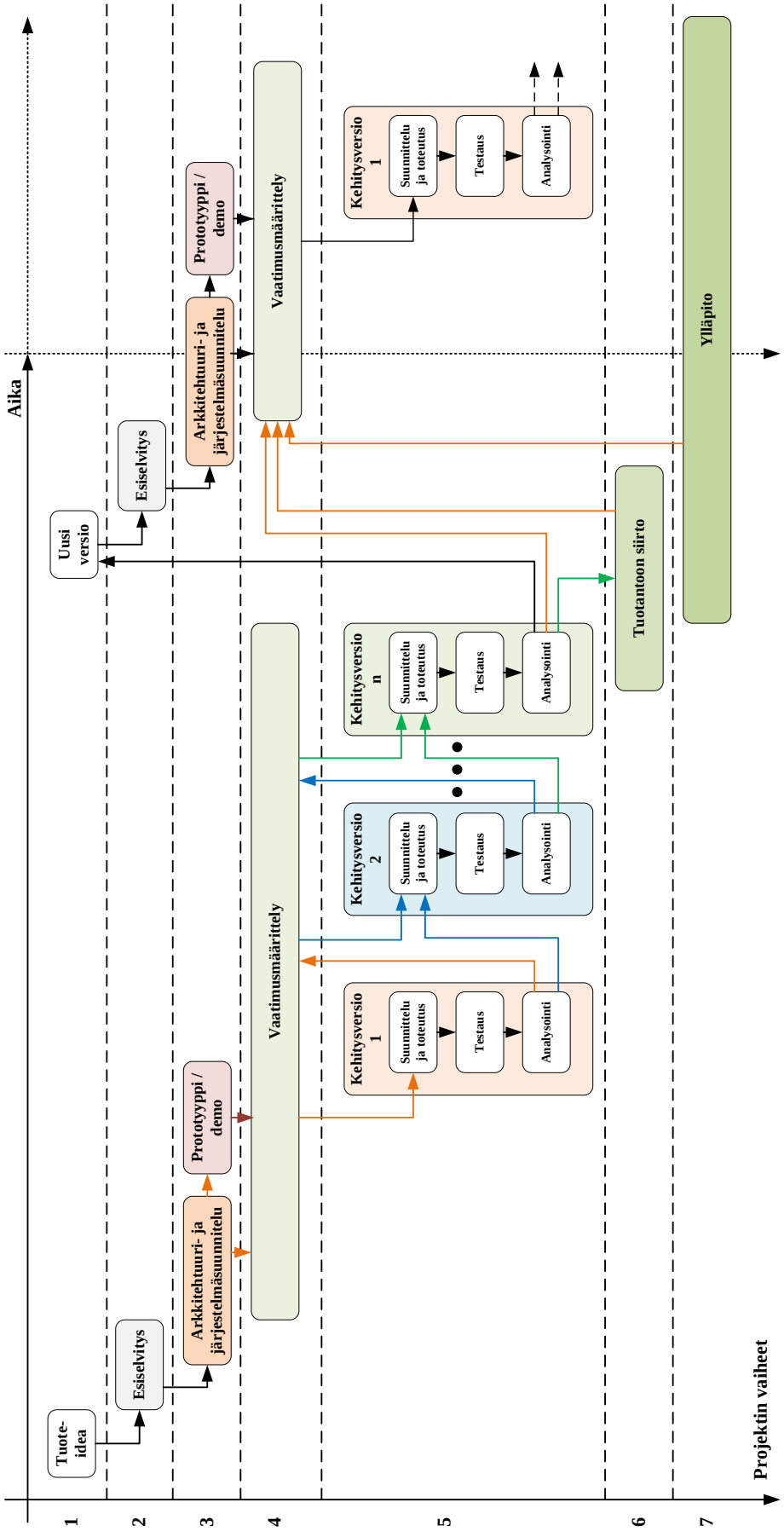
Myös uudet asiakasvaatimukset voidaan ottaa huomioon laitteistokehityksen kuluessa. Prototyyppien voidaan ajatella olevan IID-menetelmissä käytettyjen iteraatioiden tuotoksia. Jos projektin aikataulu ja resurssit sallivat, voidaan kehitysversioiden lukumäärää kasvattaa, sekä tuoda uudet ominaisuudet ja toiminnallisuudet testattavaksi niiden kautta. Laitteiston arkkitehtuuriin tai oleellisiin laitteistovaatimuksiin jälkikäteen tehtävät muutokset saattavat kuitenkin aiheuttaa projektin palaamisen vaiheeseen 2 tai 3, sekä aikataulun ja resurssitarpeen uudelleen arviointiin.

Suunnittelu-, toteutus- ja testausvaiheen aikana laitteistosta tehdään riittävä määrä kehitysversioita. Kehitysversioiden lukumäärä riippuu kehitettävän laitteen ominaisuuksista, saatavilla olevista resursseista ja esimerkiksi käytettävissä olevien valmiiden komponenttien saatavuudesta.

Kehitysversioista saadun kokemuksen ja tiedon perusteella tarpeeksi kypsä versio voidaan testata vaatimusmäärittelyä vastaan, siirtää tuotantoon ja ylläpitää sitä valmiin tuotteen elinkaaren ajan.

Uuden version tuotekehitys voidaan aloittaa usein jo ennen meneillään olevan kehitysversion siirtoa tuotantoon.

Kuva 6 havainnollistaa edellä esitettyjä tuotekehitysprojektin vaiheita ja projektin etenemistä.



Kuva 6: **Esimerkki elektroniikan laitteistokehitysprosessista**

3.1 Ketterä laitteistokehitys

Agile-menetelmien soveltamisesta laitteistokehitykseen löytyi joitain esimerkkejä lähinnä lyhyiden artikkeleiden ja mielipidekirjoitusten muodossa. Aihetta sivutaan myös kirjoissa jotka käsittelevät optimoitua tuotekehitystä (*Lean Product Development*), vaikkei näissä suoraan vertaillakaan ohjelmistokehityksessä käytettyjä menetelmiä laitteistokehitykseen.

Varsinaista prosessia tai tuotekehityksen viitekehystä elektroniikan ketterään laitteistokehitykseen en lähdehakujen yhteydessä löytänyt.

Kaikki löytämäni artikkelit nostavat esille laitteistokehitykseen liittyvät odotusajat tai kustannukset, kun laitteistokehityksen menetelmiä verrataan ohjelmistokehityksen Agile-menetelmiin (Johnson 2011; Mier 2014; Myllerup 2011; Keenan 2014; Van Schooenderwoert 2015; Graves 2014).

Laitteiston prototyyppien kustannustehokkaaseen ja nopeaan rakentamiseen ratkaisuja ovat mm. valmiiden kehitys- ja tutustumissarjojen käyttö, prototyyppien valmistukseen erikoistuneiden piirilevytoimittajien hyödyntäminen sekä mekaniikkaosien 3D-tulostus. Nopeasti tehtävissä olevat prototyypit tukevat Agile-toimintamallia, jossa lyhyehkön iteraation jälkeen jokin osakokonaisuus on valmis demonstroitavaksi tai liitettäväksi muuhun järjestelmään.

Prototyyppikierrosten ja erilaisten testiversioiden lukumäärän kasvattaminen nostaa kuitenkin tuotekehitysprojektin kokonaiskustannuksia. Usein myös esimerkiksi nopeasti saatavissa olevat piirilevyt tai koneistetut mekaniikkaosat maksavat lähtökohtaisesti enemmän kuin massatuotantoon tarkoitetuilla työkaluilla valmistetut osat.

Prototyyppikierrosten lukumäärän kasvattaminen ja eri lohkojen toiminnallisuuksien kokeileminen käytännössä on kuitenkin hyödyllistä, koska testausta päästään tekemään aikaisemmin ja tuotekehitysaikaa voidaan säästää sitä kautta. Aikaisemmin ja pienemmistä kokonaisuuksista löydetyt virheet on helpompaa ja edullisempaa korjata.

Laitteistokehitysprojektissa tulisikin löytää sopiva kompromissi prototyyppien lukumäärän, niille haluttujen toiminnallisuuksien sekä kokonaiskustannusten ja aikataulun välille.

Eric Graves on vuonna 2014 julkaistussa kirjoitussarjassaan ”*Applying Agile to Hardware Development*” analysoinut Agile-julistuksen mukaisten arvojen ja toimintatapojen sovellettavuutta laitteistokehitykseen. Yksi ohjelmistojen Agile-kehityksessä korostettu toimintatapa, jonka Graves nostaa erityisesti esiin, on uusien versioiden toimittaminen asiakkaalle asti usein ja säännöllisin väliajoin. Tällä toimintatavalla pystytään ohjelmistoon lisäämään asiakkaalle lisäarvoa tuottavia ominaisuuksia ja tekemään korjauksia käyttöönoton jälkeen havaittuihin ongelmiin. Ohjelmiston hyvin hoidettu ylläpito ja päivitykset pitävät asiakkaan tyytyväisenä ja houkuttelevat uusia asiakkaita.

Ohjelmistokehityksessä uuden version julkaisemisen kustannukset ovat yleensä maltilliset. Uuden version ohjelmistokehitykseen, testaukseen ja dokumentointiin kuluva työmäärä aiheuttaa joitain kustannuksia, mutta automatisoitu testaus ja valmiin ohjelman jakelu tai asiakkaalle siirto ei välttämättä aiheuta juurikaan lisäkustannuksia.

Laitteistokehityksessä tätä toimintatapaa on käytännössä mahdotonta toteuttaa samalla tavalla kuin ohjelmistokehityksessä. Elektroniikan ja mekaniikan tuotekehitys-, työkalu- ja testauskulut tulevat tässä esteeksi. Tuotekehitys-, työkalu- ja testauskustannukset pitäisi projektissa pystyä kattamaan myytyjen tuotteiden tuotoilla jossain järkevässä ajassa. Jos takaisinmaksuaika on pitkä, saattaa myytävän tuotteen markkinat tai komponenttien saatavuus olla muuttuneet niin paljon, että vanhan tuotteen päivittäminen tai uusien ominaisuuksien lisääminen siihen ei ole edes järkevää.

Laitteen arkkitehtuurin tekeminen modulaariseksi ja laajennettavaksi voi joissain tapauksissa pidentää laitteen elinkaarta. Modulaariseen järjestelmään voi tarjota päivityksiä ja uusia ominaisuuksia. Laitteiston modulaarisuus toisaalta kasvattaa järjestelmän hintaa ja kehityskustannusten takaisinmaksuaikaa, koska järjestelmään on suunniteltava sisään varauksia ja kapasiteettia laajennuksille.

Usein myös laitteen elektronikassa tehty muutos tai laajennusosa voi aiheuttaa laitteen tai järjestelmän uudelleen sertifioinnin eri maantieteellisille alueille. Myös sertifiointiin kuluva aika ja kustannukset tekevät uusien laitteistoversioiden julkaisemisen tiheällä aikataululla usein kannattamattomaksi.

Kolmas usein esille tulevista vertailuista Agile-menetelmien ja laitteistokehityksen menetelmien välillä on tuotteen tarpeellisten ominaisuuksien tarkka harkinta. Nk. hukan minimointi tai asiakkaalle lisäarvoa tuottamattomien ominaisuuksien pitäminen minimissään on hyödyllistä myös laitteistokehityksessä, koska sillä saadaan tehtävän työn määrä ja virhemahdollisuudet minimoitua.

Agile-julistuksen kolme ensimmäistä periaatetta ovat siis hankalia soveltaa laitteistokehitykseen. Nämä periaatteet olivat:

Noudatamme seuraavia periaatteita:

- *Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.*
- *Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.*
- *Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.*
- *...*

(Beck jne. 2001.)

Soveltamisen hankaluus tulee lähinnä laitteistokehityksen ja valmistuksen kustannuksista ja odotusajoista. Näiden takia prototyyppikierrosten ja uusien laiteversioiden välillä kuluu huomattavasti pidempi aika kuin ohjelmistokehityksessä.

Toisaalta taas loput Agile-julistuksen periaatteet ovat sovellettavissa myös laitteistokehitykseen.

Noudatamme seuraavia periaatteita:

- *...*
- *Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.*
- *Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.*
- *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*
- *Toimiva ohjelmisto on edistymisen ensisijainen mittari.*
- *Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.*
- *Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.*
- *Yksinkertaisuus – tekemättä jätettävän työn maksimointi – on oleellista.*
- *Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itse organisoituvissa tiimeissä.*
- *Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.*

(Beck jne. 2001.)

Nämä periaatteet ratkovat yleisiä ongelmia projektin tiedonkulussa ja työntekijöiden motivoinnissa.

Agile-periaatteista seitsemäs, eli ”*Toimiva ohjelmisto on edistymisen ensisijainen mittari.”* pitää laitteistokehityksen yhteydessä tosin ajatella käsittämään esimerkiksi yhtä toimivaa ja testattua lohkoa tai toiminnallisuutta. Prototyyppien rakentamisajan ja -kustannusten takia valmista laitteistoa ei pysty toimittamaan jokaisen lohkon valmistuttua. Sen sijaan eri lohkojen, osien tai toiminnallisuuksien tekeminen valmiiksi ja testaaminen irrallaan on hyödyllistä, koska silloin pienestä kokonaisuudesta voidaan löytää virheet ennen sen saamista testattavaksi koko järjestelmän mukana.

3.2 Mielipiteitä elektroniikan tuotekehityksestä

Opinnäytetyön taustatiedoiksi keräsin mielipiteitä ja kokemuksia työyhteisöni elektroniikkasuunnittelijoilta heidän mielestään hyvistä ja toimivista suunnittelukäytännöistä.

Mielipiteiden ja kokemusten keräämisessä käytin kysymyksiä eri aihealueista, jotka oman kokemukseni mukaan aiheuttavat eniten ongelmia.

3.2.1 Haastattelukysymykset

Haastattelukysymykset olivat jaettu neljään osa-alueeseen ja olivat seuraavat:

1. Vaatimusmäärittelyt
    a. Millaiset menetelmät ovat olleet käytössä elektroniikan vaatimusmäärittelyn koostamiseen?
    b. Millaisia työkaluja tai dokumentteja vaatimusmäärittelyn tekemiseen on käytetty?
    c. Miten tuotekehityksen aikaiset vaatimusten muutokset on saatu siirrettyä olemassa olevaan tuotteeseen tai prototyyppiin?

2. Aikataulutus
    a. Mikä mielestäsi määrää elektroniikan tuotekehityksen aikataulun?
    b. Millä menetelmillä elektroniikan tuotekehitystä voisi nopeuttaa?

3. Elektroniikan testaaminen
    a. Missä vaiheessa tuotteen kehitystä elektroniikan testitapausten suunnittelu on aloitettu?
    b. Millaisia apuvälineitä on ollut käytössä testaamisen nopeuttamiseen ja helpottamiseen?

4. Dokumentaatio
    a. Millaisia työkaluja tai dokumenttipohjia on ollut käytössäsi piirikortin dokumentaatiota tehtäessä?
    b. Millainen vanhan järjestelmän dokumentaatio auttaa parhaiten uutta työntekijää tai uuden tuotteen suunnittelussa?
    c. Mitä elektroniikan piirikortista tulisi dokumentoida?

3.2.2 Haastateltavien työkokemus

Opinnäytetyön taustatiedoiksi keräsin mielipiteitä ja kokemuksia työyhteisöni elektroniikkasuunnittelijoilta heidän mielestään hyvistä ja toimivista suunnittelukäytännöistä.

Elektroniikka- ja järjestelmäsuunnittelijoita, jotka ovat päivittäin tekemisissä elektroniikan tuotekehityksen kanssa, on GE Healthcare Finland Oy:llä noin 20-25 kpl.

Suunnittelijoiden työkokemus elektroniikka-alalta on keskimäärin 16 vuotta.

Työkokemusta on kertynyt eri teollisuuden aloilla toimivista yrityksistä. Eniten työkokemusta on kertynyt esimerkiksi seuraavien yritysten palveluksessa:

- Espotel Oy
- Etteplan Oyj
- GE Healthcare Finland Oy
- Microsoft Mobile Oy
- Nokia Oyj
- Philips Oy Healthcare
- Planmeca Oy
- Polar Elektro Oy
- Texas Instruments Finland Oy
- Vaisala Oyj

4 AGILE-MENETELMIEN HYÖDYNTÄMINEN LAITTEISTOKEHITYKSESSÄ

Seuraavissa luvuissa on kerättynä ajatuksia useasta elektroniikan suunnitteluun liittyvästä toiminnasta.

Ajatuksia ja hyväksi havaittuja toimintamalleja on kerätty löytyneestä kirjallisuudesta ja GE Healthcare Finland Oy:n suunnittelijoilta.
Haastatelluilla suunnittelijoilla on taustaa ja kokemusta elektroniikan tuotekehityksestä useilta vuosilta ja useiden yritysten palveluksesta.

Haastattelujen ja keskustelujen aiheet liittyivät kysymyksiin, jotka on listattu luvussa 3.2.1.

Hyväksi havaittuja menetelmiä on verrattu Agile- ja IID-menetelmissä vallitseviin käytäntöihin ja pohdittu niiden yhteneväisyyksiä ja eroja.

4.1 Vaatimusmäärittelyt

Etukäteen tehtävät vaatimusmäärittelyt pitäisi Agile-menetelmien oppien mukaisesti pitää minimissään. Vaatimusmäärittelydokumenttien sijaan Agile-menetelmissä korostetaan ihmisten välistä kommunikaatiota vaatimusmäärittelyjen siirrossa varsinaiseen toteutukseen. Kirjallisten suurien etukäteen kirjoitettujen vaatimusmäärittelyjen sijaan menetelmät korostavat nk. tuoteomistajan (*Product Owner*) läsnäoloa, jolta vaatimuksia voi kysyä suullisesti, sekä erilaisten kaavioiden ja piirrosten käyttöä suullisen kommunikaation tukena.

Koko järjestelmän tason vaatimukset voi nykypäivän sulautetuissa järjestelmissä kirjoittaa esimerkiksi lohkokaavion muotoon, koska useimmiten järjestelmä rakennetaan komponenteista, joissa käytetään vakiintuneita rajapintoja. Lohkokaavio osoittaa helposti liityntöjen tyypit (I^2C, I^2S, USB, SPI, DDR3 jne.) eikä niiden tarkempia vaatimuksia kannata toistaa erillisiksi vaatimusmäärittelyiksi. Lohkokaavion sisään voi myös kirjoittaa oleellisimpia sähköisiä vaatimuksia esimerkiksi ulkopuolisista liitynnöistä. Lohkokaavion voisikin katsoa vastaavan ohjelmistokehityksessä käytettäviä luokkakaavioita.

Erilaiset tilakoneet ja tapahtumasekvenssit on myös järkevää kuvata niille sopivilla kaavioilla sekä laitteisto- että ohjelmistokehityksessä.

Elektroniikan tuotekehityksessä vaatimusten kirjoittaminen sisään esimerkiksi eri lohkojen testitapauksiin testien hyväksyntärajojen muodossa voi myös korvata perinteisiä etukäteen kirjoitettuja vaatimuslistoja. Tätä voisi verrata ohjelmistokehityksen Agile-menetelmissä käytettäviin Test Driven Development (TDD) periaatteisiin.

Laitteistokehityksessä elektroniikan osalta asiakkaan vaatimukset ovat yleensä selkeitä ja yksiselitteisiä. Esimerkiksi elektroniikan tulee liittyä olemassa olevaan järjestelmään tai laitteeseen tietyn rajapinnan kautta, piirikortille on tarjolla tarkasti määritelty käyttöjännite tai akkukäyttöisen

laitteen toiminta-ajan tulee olla riittävä. Tällaiset vaatimukset on helppo kirjoittaa etukäteen eikä niiden tulkinnassa tule ongelmia.

Laitteiston mekaniikan suunnittelussa taas asiakkaan vaatimukset voivat olla vaikeampia kirjoittaa erilliseksi vaatimusmäärittelyksi. Erityisesti Agile-menetelmien periaate jatkuvasta kommunikoinnista tuotteen tilaajan kanssa on hyödyllistä kun tuotekehityksen aikana iteroidaan sopivaa laitteen ulkonäköä ja muotoa. Nykyiset 3D tulostusmahdollisuudet auttavat mekaniikan suunnitteluratkaisujen kommunikointia asiakkaan kanssa, koska mekaniikkamallien saaminen nopeasti ja edullisesti käsin kosketeltavaksi on mahdollista.

Haastateltujen suunnittelijoiden mielipiteet olivat melko yhtenäiset vaatimusmäärittelyjen tekemisen osalta. Laitteistokehityksessä vaatimusten pitäminen avoinna muutoksille on samaan tapaan järkevää kuin ohjelmistokehityksessäkin. Laitteiston ulkoiset rajapinnat ja arkkitehtuurin pääpiirteet on mahdollista määritellä etukäteen, mutta vaatimusten yksityiskohdat selviävät sitä mukaan kun kehitystyöryhmä oppii tilaajan palautteesta, rakennetuista prototyypeistä ja niiden testauksen tuloksista.

Projekteissa on usein määritelty yhdeksi virstanpylvääksi hetki, jolloin vaatimusmäärittely jäädytetään. Projektin aloituksesta vaatimusten jäädyttämiseen pitäisi pitää sisällään siis lähes kaikki suunnittelu ja testaus, että vaatimusmäärittely oikeasti kuvaisi tuotteen ominaisuudet.

Vaatimusten jäädyttäminen ennen varsinaisen suunnittelutyön aloittamista on mahdollista vain kaikkein yksinkertaisimmissa laitteissa.

4.2 Aikataulutus

Agile- ja IID-menetelmien yksi perusperiaatteista on nk. *time boxing*, eli tekemisen jakaminen ennalta määriteltyjen lyhyehköjen ajanjaksojen pituisiksi.

Ohjelmistokehityksessä iteraatioiden pitäminen lyhyehköinä mutta kuitenkin uusien toiminnallisuuksien toteuttaminen valmiiksi asti niiden sisällä on helpompaa kuin laitteistokehityksessä. Syy tähän on lähinnä siinä, että kun ohjelmiston julkaisuympäristö ja testaus on saatu automatisoitua, ei uuden version julkaisuun kulu aikaa minuutteja tai tunteja kauempaa.

Elektroniikan tuotekehityksen tahdin määrää usein käytettävissä olevat resurssit ja pääasiassa raha. Riippuen projektin budjetista, voidaan siihen sisällyttää tietty määrä prototyyppikierroksia ja erikoisosaamista vaativien suunnittelijoiden työaikaa.

Prototyyppikierrosten määrän lisääminen antaa enemmän vapautta toteuttaa muutoksia ja parannuksia elektroniikan piirikorttiin, mutta jokainen prototyyppikierros kasvattaa kustannuksia.

Piirilevyjen suunnittelutyö, erilaiset simuloinnit ja piirilevyn sovittaminen mekaniikkaan ovat esimerkkejä työtehtävistä, joissa useissa organisaati-

oissa on vain rajattu määrä osaajia ja joiden työaikaa jaetaan useampien projektien kesken. Erikoisosaajien vapautumista voi joutua odottamaan usein pitkäänkin.

Laitteistoläheisen ohjelmakoodin kirjoittaminen on usein myös erikois-osaamista vaativa alue. Prototyyppien elektroniikan testaamiseen tarvitaan usein ohjelmistokehittäjää, joka pystyy luomaan ohjelmakoodia elektroniikan eri lohkojen saamiseksi toimintaan niiden testausta varten. Laitteistoläheisen ohjelmakoodin erikoisosaajaan vapautumista joutuu usein odottamaan, koska nämä henkilöt ovat usein toteuttamassa myös laitteiston lopullista ohjelmistoa.

Lisäksi prototyyppikierrosten määrää ei voi kasvattaa rajattomasti, koska prototyypin hankkimiseen kuluvaa aikaa ei voi pienentää loputtomasti. Tyypillisesti elektroniikan piirilevyn hankintaan kuluvan aika on noin 3-6 viikkoa. 3-6 viikon toimitusajalla piirikortin kustannukset pysyvät vielä kohtuullisina, koska piirilevyn valmistajalle, komponenttien hankintaan ja piirilevyn ladontaan on riittävästi aikaa.

Haastatteluissa nousi esiin päällimmäisenä uuden tuotteen konseptin, arkkitehtuurin ja komponenttivalintojen eteen tehdyn ennakkotyön määrän vaikutus projektin aikatauluun.

Jos esimerkiksi mikropiirivalmistajilta on ollut saatavilla sopivia sovellusesimerkkejä, on lähes koko järjestelmä voitu rakentaa niiden avulla toimivaksi prototyypiksi. Tällaisen erillisistä piirikorteista ja sovellusesimerkeistä koostetun prototyypin kanssa myös laitteistoläheisen ohjelmakoodin kirjoitus on voitu aloittaa ajoissa, jolloin kun ensimmäinen varsinainen prototyyppi saadaan valmiiksi, on sen testaukseen tarvittavat työkalut ja menetelmät jo valmiina. Konseptiehdotus ei pitäisikään olla pelkkä ehdotus pääkomponenteista ja hahmotelma järjestelmän lohkokaaviosta. Konseptien suunnittelussa tulisi ottaa tavoitteeksi tehdä toimiva prototyyppi, että edes jotain ohjelmakoodia on valmiina millä eri lohkojen toiminta on todettu etukäteen.

Varsinaisen tuotekehitysprojektin ja esitutkimuksen erottaminen toisistaan on siis hyvä tapa, koska esitutkimuksessa on usein helpompaa kokeilla erilaisia lähestymistapoja ja hylätä toimimattomat ratkaisut. Jos huono ratkaisu pääsee lopputuotteen kehitykseen tähtäävään projektiin asti, on aikataulu- ja kustannuspaineen alla houkutus vian paikkaamiseen vain jotenkuten toimivaksi liian suuri.

Toinen haastatteluissa esiin usein tullut asia oli kehitystyöryhmän osaaminen ja oleellisten resurssien puuttuminen eri vaiheissa. Erityisesti matalan tason ohjelmiston ja laiteajurien saatavuus oli useissa tapauksissa viivästyttänyt projektia, koska tarvittavaa ohjelmistoa ei ollut saatavilla testauksen aloittamiseksi. Ratkaisuna tähän ongelmaan esitettiin sitä, että elektroniikan tuotekehityksen työryhmään pitäisi aina kiinnittää 1-2 ohjelmistokehittäjää vastaamaan testattavuudesta yhdessä elektroniikkasuunnittelijoiden kanssa.

Toinen osaamisalue, joka tuli esille ja jossa on usein puutteita, on piirilevyjen ja elektroniikan simulointi yhdessä. Elektroniikan perussimulointeihin ei useinkaan ole saatavilla piirilevyn malleja, koska tähän tarkoitukseen tarvittavaa työkalua ja sen käytön osaavaa henkilöä ei ole ollut saatavilla. Elektroniikan suunnittelussa on pitänyt mennä niin sanotusti sokkona prototyypin valmistumiseen asti, että on päässyt toteamaan toimiiko jokin kytkentä käytännössä. Testauksessa ilmenneet ongelmat olisivat voineet paljastua jo simuloinneissa, eikä prototyypin rakentamiseen kulunutta aikaa olisi hukattu.

Laitteistokehityksessä olisi siis hyödyllistä noudattaa Agile-menetelmien periaatetta työryhmän itseohjautuvuudesta ja riittävästä osaamisesta. Työryhmälle annettu vastuu jonkun kokonaisuuden toteuttamisesta valmiiksi asti motivoi työntekijöitä ja tuottaa yleensä parempia ratkaisuja kuin työn tekeminen erillisten yksilöiden suorittamina erillisinä tehtävinä. Ohjelmistokehityksen toimintamallien tapaan laitteistokehitystyöryhmä voisi koostua 4-9 osaajasta, jotka pystyvät suunnittelemaan ja testaamaan piirikortin ja sen tarvitsemat lisälaitteet ja mekaniikan valmiiksi asti itsenäisesti.

Työryhmän riittävän kattava osaaminen ja itseohjautuvuus helpottaisivat projektin aikataulutusta, koska aikatauluun vaikuttavien asioiden tuntemus olisi aina saatavilla ja siirrettävissä työmääräarvioihin.

4.3 Elektroniikan testaus

Monen Agile-menetelmän käytäntö on jatkuva testaus. Jokaiselle ohjelmiston uudelle osalle (*unit*) tulisi kirjoittaa yksikkötesti (*unit test*). Uutta koodia integroitaessa muuhun ohjelmistoon, tulisi suorittaa integraatiotestaus. Yksikkö- ja integraatiotestauksen avulla pyritään varmistamaan uuden koodin tai muutosten toimivuus ja yhteensopivuus muun ohjelmiston kanssa.

Joissain Agile-menetelmissä korostetaan testausta jopa niin paljon, että suositellaan testien kirjoittamista ennen varsinaisen koodin kirjoittamista. Etukäteen kirjoitetut ohjelmiston hyväksyntätestitapaukset voivat toimia vaatimusmäärittelyjen tilalla kuvaamaan ohjelmiston haluttua toimintaa. Yksikkötestien etukäteen kirjoittaminen voi selkeyttää ja nopeuttaa varsinaisen ohjelmakoodin kirjoittamista, koska ohjelmistokomponentin rajapinnat ja poikkeusten käsittely pitää miettiä valmiiksi testitapausten kautta.

Agile-menetelmissä toinen pääperiaate on myös yksikkö- ja integraatiotestauksen automatisointi, jolloin uuden koodin tai koodimuutosten testaus on helppoa ja nopeaa. Yksikkötestauksen toteuttamisen automatisointi on käytännössä ainoa vaihtoehto siihen, että jo kirjoitetun ohjelmakoodin optimoinnin (*refactoring*) vaikutukset on nopea testata.

Laitteistokehityksessäkin jatkuvan testauksen periaate olisi myös hyödyllinen. Usein testaamisen aloitusta viivästää se, että piirilevyn valmistukseen ja hankintaan kuluva aika on pitkä ja prototyypin hinta rajoittaa tehtävien piirilevykierrosten määrää.

Piirikaavion erilaisille kytkennöille ja lohkoille on kuitenkin mahdollista kirjoittaa testikuvauksia etukäteen. Testauksen voi myös aloittaa jollain muulla piirilevyllä, esimerkiksi mikropiirivalmistajan tutustumissarjalla, jos kytkentään tehtävät muutokset tai vaikka piirilevyn vaikutus toimintaan ei ole oleellinen.

Seuraavissa luvuissa on esitetty esimerkkejä siitä, miten sulautetun järjestelmän laitteistoa voi testata niin, että laitteistokehityksessä päästäisiin lähemmäs IID-menetelmien yksikkötestauksen periaatteita mahdollisimman aikaisesta testauksesta ennen komponenttien integrointia kokonaisen järjestelmää.

4.3.1 Kytkentöjen simulointi

Kytkentöjen simulointien avulla pystytään helposti suorittamaan niiden verifiointia ennen kuin varsinaisen prototyypin mittauksia päästään aloittamaan.

Spice- ja RLC-mallien sekä Spice-simulointien avulla pystytään useimmiten määrittelemään esimerkiksi passiivikomponenttien DC-toimintapisteet. DC-toimintapisteiden avulla näkee, että eri komponenttien läpi kulkevat virrat, niiden yli olevat jännitteet tai komponenteissa syntyvä tehohäviö pysyy vaaditun varmuuskertoimen verran komponenttivalmistajan ilmoittamien arvojen alapuolella (*derating*).

Analogisten kytkentöjen taajuusvasteet on myös helppo selvittää etukäteen simulointien avulla. Taajuusvasteiden simuloinneissa on myös mahdollista ottaa huomioon komponenttien epälineaarisuudet, sarjaresistanssit sekä loiskapasitanssit ja -induktanssit. Esimerkiksi keraamisten kondensaattoreiden kapasitanssin muutokset jännitteen suhteen tai häiriönpoistoferriittien impedanssin muutokset virran suhteen on mahdollista ottaa huomioon etukäteen ja nähdä niiden vaikutus signaalien säröytymiseen.

Ennen prototyypin piirilevyn valmistustiedostojen luontia voidaan simulointien avulla tarkistaa signaalien ja tehonsyöttöjen laatu (*Signal Integrity (SI), Power Integrity (PI)*). Saatavilla on mallinnustyökaluja, joilla voidaan luoda esimerkiksi S-parametrimalli piirilevyn yksittäisen signaalin, väylän tai vaikka tehonsyötön kytkennöistä. Piirilevystä luotuun S-parametrimalliin voidaan liittää mikropiirivalmistajien tarjoamia Spice- tai IBIS-malleja sekä passiivikomponentteja RLC-malleina.

Eri malleista koostetun simulointimallin avulla voidaan tarkistaa esimerkiksi nopean digitaalisen signaalin impedanssisovituksen sekä tehonsyötön impedanssin ja kuormatransienttivasteen (*load transient response*).

Simulointien tulosten avulla piirilevyä ja komponenttivalintoja voidaan muuttaa ennen prototyypin tilaamista.

Simulointeja voisi verrata ohjelmistokehityksen yksikkötesteihin. Kun prototyyppi on saatu mitattavaksi, voidaan aikaisemmin simuloidut yksi-

tyiskohdat tarkistaa mittaamalla. Mittauksia voisi verrata ohjelmistokehityksen integraatiotestaukseen.

Simulointi- ja mittaustulosten vertailu auttaa kehittämään simulointityökalujen käyttöä työryhmässä.

4.3.2 Piirivalmistajien sovellusesimerkkien käyttö

Mikropiirien valmistajien sovellusesimerkkejä voidaan käyttää järjestelmän arkkitehtuurin kehityksessä. Sovellusesimerkkeihin voidaan myös esimerkiksi liittää oheislaitteita, mitata virrankulutusta tai signaalien toimintaa, sekä niiden avulla voidaan kehittää laitteistoläheistä ohjelmistoa ja testitapauksia.

Sovellusesimerkkien avulla voidaan myös rakentaa kokonainen järjestelmä tai sen osa, jos esimerkiksi olemassa olevan sulautetun järjestelmän jotain osaa halutaan muuttaa tai parantaa. Tällöin päästään testaamaan aikaisessa vaiheessa laitteistoa lähes oikealla elektroniikalla.

Agile-menetelmissä ohjelmistokehityksessä pyritään minimoimaan hukan määrää ja käyttämään olemassa olevia toimivia ratkaisuja hyväksi. Valmiista ratkaisuista on yleensä olemassa valmiita testitapauksia ja tuloksia, jotka nopeuttavat kehitystä ja antavat luottamusta ratkaisun toimivuuteen.

Piirivalmistajien sovellusesimerkkejä kannattaa käyttää laitteistokehityksessä minimoimaan tehtävän työn määrää samalla tavalla kuin valmiita avoimen lähdekoodin ohjelmistokomponentteja ohjelmistokehityksessä.

4.3.3 Erityisohjelmisto laitteistotestaukseen

Sulautetuissa järjestelmissä laitteiston liitynnät ja ohjaussignaalit on helpointa testata, jos niitä pääsee ohjaamaan tai lukemaan jonkin rajapinnan kautta.

Esimerkiksi ylimääräisen sarjaportin varaaminen testauskäyttöön nopeuttaa elektroniikan testaamista, koska prosessorin liityntöjen ohjaus matalan tason ohjelmakoodilla ja erilaisilla testijonoilla on silloin mahdollista.

Luotuja testijonoja voidaan myös käyttää samojen testien toistamiseen elektroniikan eri kehitysversioista. Jos on tiedossa että sama testi on tehtävä vähintään kaksi kertaa, on testien automatisointi yleensä kannattavaa.

Elektroniikan lohkojen testauskäyttöön tehdyt ajurit ja testausjonot voidaan yleensä myös hyödyntää lopullisen ohjelmiston kehityksessä ja tuotannon korttitestauksessa.

Laitteistotestaukseen tehtävää ohjelmistoa ja testijonoja voisi verrata ohjelmistokehityksen automatisoituun yksikkö- ja integraatiotestaukseen.

4.3.4 Testilevyt elektroniikan testauksen helpottamiseen

Elektroniikan testauksen helpottamiseen ja laitteistotestauksen erityisohjelmiston tueksi on usein kannattavaa tehdä erillinen testilevy.

Testausten suunnittelu etukäteen erillisten testilevyjen liityntöjä hyödyntäen nopeuttaa testausta.

Testilevyille voidaan rakentaa kytkentöjä joilla yksittäiset lohkot voidaan eristää testausta varten ja niiden signaaleihin pääsee käsiksi nopeasti ilman esim. juotattavia kytkentälankoja. Testilevyjen avulla lohkojen toiminnan varmistus ja muutosten vaikutusten todentaminen nopeutuu.

Esimerkiksi testiliittimien lisääminen suoraan piirilevylle tai piirilevystä ulospäin tuleville siivekkeille ja signaalien yhdistäminen testiliittimiin nopeuttaa ja helpottaa testauksen automatisointia.

Testilevyissä hyväksi havaittu tapa on pitää kehitettävän laitteiston piirilevyn muoto ja piirilevyn kerrosjako samana kuin lopulliseen versioon on ajateltu. Lisäämällä piirilevyn keskelle 2 tai 4 lisäkerrosta testiliityntöjen tekemiseen auttaa pitämään testilevyn tekemiseen kuluvan suunnitteluajan lyhyenä.

Toinen helppo tapa tehdä liitynnät testilevylle on käyttää nk. piikkipetiä ja testipisteitä. Testausta helpottamaan käytettävät liittimet voidaan järjestellä erilliselle piirilevylle, josta liitytään testipiikeillä testattavaan levyyn. Piikkipeti sopii tosin huonosti tuotekehityksen alkuvaiheeseen, koska testipisteiden paikkoja voidaan joutua muuttamaan useaan kertaan eri prototyyppikierrosten välillä.

Kerran rakennettu testausjärjestely ja testitapaukselle kirjoitettu testijono voidaan toistaa helposti, kun testilaitteiden liittäminen piirilevylle tapahtuu muutamalla liittimellä ja samalla tavalla kuin edellisellä kerralla.

4.3.5 Standardirajapintojen testaus

Sulautetuissa järjestelmissä hyödynnetään järjestelmäkomponentteina yleisesti mikropiirejä tai antureita, joissa käytetään standardiksi muodostuneita rajapintoja.

Yleisiä liityntöjä ovat esimerkiksi:

- I^2C väylä prosessorin ja esim. anturien tai EEPROM muistipiirien välille
- SPI väylä prosessorin ja esim. AD-muunninten tai Flash-muistipiirien välille
- SDIO liityntä prosessorin ja esim. WLAN radio moduulin tai muistikortin välille
- CAN ja LIN väylät erityisesti ajoneuvoympäristössä eri toimilaitteiden väliseen kommunikaatioon
- USB väylän eri versiot kuten USB 2.0, 3.0 ja IC-USB
- Muistiväylät kuten DDR2, DDR3, DDR4, LP-DDR2, LP-DDR3

- Ethernet liityntöjen eri versiot parikaapelille kuten 10 Mbps IEEE 802.3i, 100 Mbps IEEE 802.3u ja 1000 Mbps IEEE 802.3ab
- PCI Express oheislaiteväylä prosessorin ja esim Ethernet tai näytönohjaimen välille
- LVDS ja eDP (embedded display port) näyttöpaneeliliitynnät
- HDMI ja DVI-D näyttöliityntöjen TMDS signalointi
- Display Port näyttöliityntä.

Näiden liityntöjen käyttö on yleensä suoraviivaista, koska mikropiirien valmistaja on suunnitellut ja testannut piirin liitynnät standardia vasten.

Omassa laitteistossa liityntöjen testaukseen liittyy lähinnä piirilevyyn ja liityntään tai väylään liitettyjen komponenttien sähköisen toiminnan toteaminen ja testaus. Sähköiseen testaukseen on tarjolla useita erilaisia oskilloskooppeja ja valmiita ohjelmistoja sekä testijigejä, joilla liityntöjen testaaminen on suoraviivaista ja nopeaa.

Standardiliityntöjen testaus voidaankin helposti suorittaa vaikka jokaisesta prototyyppikierroksesta, jos käytössä on sopivat työkalut.

Standardirajapintojen testausmäärittelyt ja niille rakennetut testausjonot ovat erittäin helposti uudelleenkäyttävissä eri laitteiden kanssa. Jos yrityksessä on hankittuna tarpeelliset mittalaitteet testi-jigeineen, voi kerran kirjoitettua testitapausta hyödyntää kaikkien laitteistojen testauksessa ja näin mahdollistaa testaus jokaisen muutoksen jälkeen.

4.4 Dokumentaatio

Agile-menetelmissä ja -julistuksessa korostetaan turhan dokumentaation jättämistä pois ja dokumentaation kirjoittamista oikealla hetkellä projektissa.

Myös elektroniikan suunnittelussa dokumentaation määrää voidaan pienentää, kun ymmärretään mitä normaalisti tuotetuissa dokumenteissa on jo lähtökohtaisesti sisällä.

Elektroniikassa kolme oleellisinta dokumentaatiota ovat:

1. piirikaavio
2. osaluettelo
3. piirilevyn valmistustiedostot.

Nämä kolme dokumenttia toteutetaan yleensä projektissa automaattisesti, että saadaan piirilevy tuotettua valmiiksi asti. Näissä dokumenteissa on sisään kirjoitettuna paljon oleellista tietoa.

Piirikaavio ja osaluettelo määrittelevät liitinten tyypit ja nastajärjestyksen, joten näiden kopioimista erilliseen vaatimusmäärittelyyn voidaan pitää turhana.

Piirilevyn valmistustiedostot pitävät sisällään liitinten paikoituksen, joten niidenkään erikseen määritteleminen erilliseen vaatimusmäärittelyyn ei ole

tarpeen. Liitinten ja korkeiden komponenttien paikoitus piirilevylle tulee usein laitteen mekaniikan ja piirilevyn suunnittelun iteraatioista, joten näiden komponenttien paikoituksen etukäteen määritteleminen ei ole yleensä muutenkaan mahdollista.

Riippuen käytettävistä piirikaavion ja piirilevyn suunnitteluohjelmista ja komponenttikirjastosta, voi tiedostot pitää sisällään myös esimerkiksi komponenttien 3D-mallit ja erillisten kytkentöjen tyyppien ja riippuvuuksien määrittelyt (väylän signaalit ja sallitut erot viipeissä, piirilevyvetojen ja differentiaaliparien impedanssit, sallitut reitityskerrokset jne.). Näidenkin yksityiskohtien kopioiminen toiseen dokumenttiin tai piirrokseen voi useimmiten jättää pois.

Kokenut suunnittelija näkee edellä mainituista kolmesta dokumentista melko pitkälle miten erilaiset kytkennät toimivat, mutta yksityiskohdat joilla esimerkiksi erilaiset passiivikomponenttityypit tai niiden arvot on valittu, ei pelkästä piirikaaviosta näy.

Lisäksi elektroniikan toiminnallisuuden kommunikoiminen ohjelmistokehittäjille voi olla vaikeaa pelkän piirikaavion avulla. Suunnitelturatkaisujen ja toiminnallisuuksien kommunikointiin toimivaksi tavaksi on todettu kaksi lisädokumenttia – elektroniikan toimintakuvaus ja lohkokaavio.

4.4.1 Lohkokaavio

Toimintakuvauksen johdantona tai piirikaavion etulehtenä voidaan käyttää lohkokaaviota. Lohkokaavio voi olla myös erillinen dokumentti.

Lohkokaaviosta tulisi näkyä kaikki oleelliset toiminnallisuudet omina lohkoinaan, sekä lohkojen väliset rajapinnat.

Lohkokaaviota voisi pitää toimintakuvauksen ja piirikaavion visuaalisena sisällysluettelona, josta on helppo viitata järjestelmän eri osiin tai piirikortin eri toiminnallisuuksiin.

4.4.2 Toimintakuvaus

Piirilevyn valmistustiedostojen ja piirikaavion lisäksi piirilevystä olisi hyvä tuottaa toimintakuvaus, joka perustelee eri lohkojen komponenttivalinnat, oleelliset suunnitteluparametrit ja esim. piirilevyn rakenteen ja eri kerrosten käyttötarkoituksen.

Toimintakuvaukseen on myös järkevää liittää sulautetun järjestelmän ohjelmiston tai käyttöjärjestelmän ajuritason rajapintojen kuvaukset.

Vuokaaviot ja tilakaaviot erilaisten signaalien käsittelylle sekä eri mikropiirien osoiteavaruudet ja alustukset helpottavat ohjelmiston kirjoittamista ja järjestelmän ylläpitoa ja päivityksiä.

Järjestelmän käyttöjännitteiden ja ohjaussignaalien toiminta laitteiston käynnistyksessä ja sammutuksessa on hyvä kuvata erillisellä käynnistys ja sammutus sekvenssillä.

Toimintakuvaus tulisikin kirjoittaa niin, että piirikaavion, osaluettelon, piirilevyn valmistustietojen sekä toimintakuvauksen perusteella

- järjestelmän ja lohkojen eri signaalien oleelliset vaatimukset tulevat esille
- valmiin järjestelmän ylläpitäjä pystyy tekemään muutoksia elektroniikkaan konsultoimatta alkuperäistä suunnittelijaa
- suunniteltujen ja testattujen lohkojen sekä kytkentöjen uudelleenkäyttö onnistuu piirilevyn seuraavassa versiossa tai toisessa tuotteessa
- aloittelevat suunnittelijat voivat perehtyä käytettyihin kytkentöihin itsenäisesti
- uuden laitteistoversion suunnittelijat ymmärtävät laitteiston rajoitukset ja ominaisuudet joita ei toteutettu alkuperäisessä projektissa.

Ohjelmistokehityksessä dokumentaatio on usein ohjelmistoa käsittelevässä Wikissä sen sijaan että ylläpidettävänä olisi erillinen dokumentti. Lisäksi ohjelmakoodiin on tapana kirjoittaa kommentteja, jotka selvittävät koodia lukevalle yksityiskohtia toteutetuista ratkaisuista.

Samanlainen tapa on osoittautunut toimivaksi myös elektroniikan toimintakuvauksen kirjoittamisessa. Wikin avulla dokumentaation pitäminen kevyenä ja ylläpidoltaan hajautettuna tukee Agile-menetelmien ajatusta dokumentaatiosta, joka on tehty vain tarpeeseen ja mahdollisimman kevyenä. Piirikaavion yhteyteen kirjoitetut pienet kommentit auttavat piirikaaviota lukevaa henkilöä ymmärtämään vaikkapa jonkin komponentit piirilevylle sijoittelun merkitystä tai oleellisen suunnitteluparametrin valintaa.

Lohkon toimintakuvauksen kirjoittamisen voi tehdä myös toinen suunnittelija esimerkiksi kirjoittaessaan lohkolle testejä tai katselmoidessaan piirikaaviota. Tällä tavalla alkuperäisen piirikaavion suunnittelija ja kuvauksen kirjoittaja joutuvat pakosti kommunikoimaan oleelliset asiat toimintakuvauksen kirjoittamiseksi ja tietotaito suunnittelutiimissä siirtyy.

Tätä toimintatapaa voisi verrata Agile-ohjelmistokehityksessä käytettävään tapaan järjestää koodikatselmoinnit kaikelle tuotettavalle ohjelmakoodille, tai vaikka XP-menetelmässä käytettävään pariohjelmointiin.

5 YHTEENVETO JA JOHTOPÄÄTÖKSET

Opinnäytetyössä tutustuttiin jatkuvan ja lisäävän kehityksen periaatteisiin sekä kolmeen ohjelmistokehityksen menetelmään, joissa sovelletaan näitä periaatteita. Lisäksi tutustuttiin ketterän ohjelmistokehityksen julistukseen.

Saatavilla olevasta kirjallisuudesta etsittiin myös jo dokumentoituja menetelmiä tai ajatuksia ohjelmistokehityksen ketterien menetelmien soveltamisesta elektroniikan suunniteluun.

Lisämateriaaliksi haastateltiin tekijän työyhteisön elektroniikkasuunnittelijoita heidän hyviksi kokemien menetelmien ja käytäntöjen keräämiseksi opinnäytetyön taustatiedoiksi.

Opinnäytetyön johtopäätös on, että ohjelmistokehityksen ketteriä menetelmiä voidaan hyödyntää sulautettujen järjestelmien laitteistokehityksessä, mutta ei aivan sellaisenaan.

Ohjelmistokehityksen tapa jaksottaa kehitys sopivan pituisiin jaksoihin ja julkaista uusi versio jakson päätteeksi vaatii laitteistokehityksen yhteydessä erilaista toimintatapaa. Agile-menetelmiä sovellettaessa laitteistokehitykseen on huomioitava laitteiston prototyyppien rakentamiseen liittyvät odotusajat ja kustannukset.

Kuitenkin vesiputousmalliin perustuvan tuotekehitysprojektin korvaaminen jatkuvan ja lisäävän kehityksen periaatteilla olisi hyödyllistä myös laitteistokehityksessä. Vaikka valmista toimivaa laitetta ei pystyttäisikään toimittamaan jokaisen tuotekehitysiteraation lopuksi, auttaa työn jakaminen vaikkapa valmiiden lohkojen toimittamiseen jaksoissa pitämään kerralla työn alla olevien tuntemattomien muuttujien määrän minimissään. Myös nykypäivänä saatavilla olevat valmiit sovellusesimerkit mikropiireistä ja 3D-tulostuspalvelut auttavat nopeuttamaan prototyyppien rakentamista ja nopean palautteen saamista kehitettävästä laitteesta.

Toisaalta iso osa ohjelmistokehityksen ketteristä menetelmistä voidaan soveltaa melko suoraviivaisesti ja ne voisivat auttaa myös laitteistokehityksessä.

Dokumentaation pitäminen kevyenä, sen tekeminen joustavasti muun suunnittelutyön ohessa sekä käyttäen esim. Wiki-tyylistä työkalua voisi olla hyödyllistä myös laitteistokehitysprojektissa. Laitteistokehitysprojektissa usein esiintyy pienimuotoinen paniikki projektin lopussa, kun huomataan että iso osa dokumentaatiosta on kesken. Kevyt dokumentaatio tehtynä aina jonkin lohkon osalta valmiiksi asti iteraatioiden kuluessa voisi poistaa tällaiset yllätykset projektissa.

Dokumentaatiossa on myös hyödyllistä kirjoittaa se silmälläpitäen uudelleenkäytettävyyttä ja työnkiertoa. Työssä selvisi että useat suunnittelijat pitävät elektroniikan lohkojen toimintakuvausta tärkeänä dokumenttina, joka tukee nimenomaan jo tehdyn työn hyödyntämistä uusissa projekteissa uusien suunnittelijoiden voimin.

Vaatimusmäärittelyn täydentäminen joustavasti projektin aikana etukäteen kirjoitetun sijaan olisi myös hyödyllistä. Esiselvityksen, käyttäjän tarpeiden ja laitteiston arkkitehtuurin kehityksen aikana saadaan usein riittävä määrä vaatimuksista kootuksi, että projekti voidaan aloittaa. Mutta yksityiskohtaisten vaatimusten etukäteen määrittäminen on usein hukkaan heitettyä työtä. Liian tarkka vaatimusmäärittely voi haitata parhaan ratkaisun löytymistä, eikä tarkkoja vaatimuksia voida arvata etukäteen ennen lohkon tai toiminnallisuuden toteutusta ja sen testaamisesta saatua kokemusta.

Tilaajan ja työryhmän yhteistyön parantamiseen tähtäävät menetelmät, kuten tilaajan edustajan jatkuva läsnäolo kehitystyöryhmän kanssa samassa tilassa sekä prototyyppien ja toiminnallisuuksien esittelyt kehityksen kuluessa voisivat toimia myös laitteistokehityksessä.

Tärkein etu tilaajan jatkuvasta läsnäolosta on nopean palautteen saaminen yksityiskohdista. Jos palautteen saaminen vie liikaa aikaa, ehtii suunnittelija viemään omaa näkökulmaansa eteenpäin ja jo tehdyn työn muuttaminen tai palaaminen jopa alkupisteeseen aiheuttaa hukkaa ja hidastaa kehitystyötä.

Testattavuuden ja testausautomaation kehittäminen on ohjelmistokehityksessä itsestäänselvyys. Useat käytössä olevat menetelmät korostavat jopa testitapausten suunnittelua ennen toteutuksen aloitusta. Myös ohjelmistojen jatkuvan integraation ja koodin jatkuvan optimoinnin takia testauksen automaatioon kiinnitetään ohjelmistokehityksessä erityistä huomiota.

Laitteistokehityksessä samojen periaatteiden soveltaminen voisi auttaa tekemään elektroniikan suunnittelusta joustavampaa. Testitapausten etukäteen määrittely voitaisiin yhdistää vaatimusmäärittelyn tekemiseen. Testausautomaation kehittäminen taas nopeuttaisi mittaustulosten saamista prototyypeistä. Automatisoitu testaus hyödyttää erityisesti silloin, kun jokin asia ei ole toiminut ensimmäisellä kerralla ja tehtyjen korjausten vaikutusta halutaan arvioida uudesta prototyypistä.

Työryhmän itseohjautuvuutta on korostettu erityisesti Scrum-menetelmässä. Kun työryhmän sisältä löytyy kaikki tarvittava osaaminen ja työkalut halutun toteutuksen tekemiseen ja työryhmä itse jakaa vastuun toimituksesta, nopeutuu kehitystyön tekeminen. Kehitystyön nopeutuminen voi johtua siitä, että työn ohjaamiseen ei tarvita erillistä esimiestä, mutta myös itseohjautuvassa työryhmässä suunnittelijoiden motivaatio voi olla parempi ja työn tehokkuus kasvaa mielekkäämmän työskentelytavan takia. Samat periaatteet voisivat toimia myös laitteistokehityksessä.

Useat ohjelmistokehityksen ketterät menetelmät voisivat siis parantaa myös laitteistokehitysprojektin onnistumisen mahdollisuuksia ja niiden soveltamista kannattaa harkita.

LÄHTEET

Basili, V. & Larman, C. 2003. Iterative and Incremental Development: A Brief History. IEEE Computer, Volume 36 47-56. ISSN: 0018-9162

Beck, K. Beedle, M. van Bennekum, A. Cockburn, A. Cunningham, W. Fowler, M. Grenning, J. Highsmith, J. Hunt, A. Jeffries, R. Kern, J. Marick, B. Martin, R. Mellor, S. Schwaber, K. Sutherland, J. & Thomas, D. 2001. Agile Manifesto. Viitattu 21.10.2016. http://agilemanifesto.org/

Beck, K. Beedle, M. van Bennekum, A. Cockburn, A. Cunningham, W. Fowler, M. Grenning, J. Highsmith, J. Hunt, A. Jeffries, R. Kern, J. Marick, B. Martin, R. Mellor, S. Schwaber, K. Sutherland, J. & Thomas, D. 2001. Principles behind the Agile Manifesto. Viitattu 21.10.2016. http://agilemanifesto.org/principles.html

Booch, G. Jacobson, I. & Rumbaugh, J. 1999. The unified software development process. Boston: Addison-Wesley Longman Publishing Co. ISBN: 0201571692

Coad, P. Lefebvre, E. & De Luca, J. 1999. Java Modeling In Color With UML. Prentice Hall. ISBN: 013011510X

Graves, E. 2014. Applying Agile to Hardware Development (Part1). Playbook. Viitattu 21.10.2016. https://www.playbookhq.co/blog/agileinhardwarenewproductdevelopment/

Graves, E. 2014. Applying Agile to Hardware Development (Part2). Playbook. Viitattu 21.10.2016. https://www.playbookhq.co/blog/agile-work-hardware-new-product-development-environments-part-2/

Graves, E. 2014. Applying Agile to Hardware Development (Part3). Playbook. Viitattu 21.10.2016. https://www.playbookhq.co/blog/applying-agile-hardware-new-product-development-part-3/

Graves, E. 2014. Applying Agile to Hardware Development (Part4). Playbook. Viitattu 21.10.2016. https://www.playbookhq.co/blog/applying-agile-hardware-new-product-development-part-4/

Graves, E. 2014. Applying Agile to Hardware Development (Part5). Playbook. https://www.playbookhq.co/blog/applying-agile-hardware-new-product-development-part-5/

Graves, E. 2014. Applying Agile to Hardware Development (Part6). Playbook. Viitattu 21.10.2016. https://www.playbookhq.co/blog/applying-agile-hardware-new-product-development-part-6/

Johnson, N. 2011. Design How-To: Agile hardware development – nonsense or necessity?. EE Times. Viitattu 21.10.2016. http://www.eetimes.com/document.asp?doc_id=1279137

Juhola, T. 2010. Customized agile development process for embedded software development – A study of special characteristics of embedded software and agile development. Saarbrücken: VDM Verlag Dr. Müller. ISBN: 9783639285956

Keenan, M. 2014. Comment: An agile approach to electronics design. Electronics Weekly. Viitattu 21.10.2016. http://www.electronicsweekly.com/news/design/comment-rs-components-agile-design-strategy-2014-09/

Larman, C. 2012. Agile & Iterative Development, A Manager's Guide. Addison-Wesley. ISBN: 9780131111554

De Luca, J. 2006. Updated FDD Overview Presentation, Nebulon Pty. Ltd. Viitattu 21.10.2016. http://www.featuredrivendevelopment.com/files/fddoverview.pdf

Mier, F. & Zorzano, J. 2014. How We Learned to Built Hardware, the Agile way. Vision Mobile. Viitattu 21.10.2016. https://www.visionmobile.com/blog/2014/02/learned-built-hardware-agile-way

Myllerup, B. 2011. Why Agile Does Matter in an Embedded Development Environment. Scrum Alliance. Viitattu 21.10.2016. https://www.scrumalliance.org/community/articles/2011/march/why-agile-does-matter-in-an-embedded-development-e

Nonaka, I. Takeuchi, H. 1986. The New New Product Development Game. Harvard Business Review. Viitattu 21.10.2016. https://hbr.org/1986/01/the-new-new-product-development-game

Reinertsen, D. 2009. The Principles of Product Development Flow: Second Generation Lean Product Development. Los Angeles: Celeritas Publishing. ISBN: 9781935401001

Van Schooenderwoert, N. 2015. Yes, Hardware Can Be Agile!. InfoQ. Viitattu 21.10.2016. https://www.infoq.com/articles/hardware-can-be-agile

Schwaber, K. Sutherland, J. 2016. The Scrum Guide™ - The Definitive Guide to Scrum: The Rules of the Game. Scrumguides.org. Viitattu 21.10.2016. http://www.scrumguides.org/download.html

Smith, P. 2007. Flexible Product Development: Building Agility for Changing Markets. San Fransisco: Jossey-Bass. ISBN: 9780787995843

Wikipedia. Feature Driven Development. Viitattu 4.8.2016. https://en.wikipedia.org/w/index.php?title=Feature-driven_development&oldid=720590071

Wikipedia. Iterative and Incremental Development. Viitattu 18.7.2016. https://en.wikipedia.org/w/index.php?title=Iterative_and_incremental_development&oldid=705841193

Wikipedia. Scrum (software development). Viitattu 4.8.2016. https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=732372599

Wikipedia. Unified Process. Viitattu 18.7.2016. https://en.wikipedia.org/w/index.php?title=Unified_Process&oldid=724140483