AATOS
-artikkelit

Erja Nikunen

# Tekemällä oppien ohjelmistoammattilaiseksi

Metropolia Ammattikorkeakoulun julkaisusarja

Erja Nikunen

# Tekemällä oppien ohjelmistoammattilaiseksi

Metropolia Ammattikorkeakoulun julkaisusarja

AATOS-ARTIKKELIT 11 • 2013

AATOS
-artikkelit

Kustantaja Metropolia Ammattikorkeakoulu

ISBN 978-952-6690-08-7
ISSN 1799-604X

# SISÄLLYS

# Johdanto

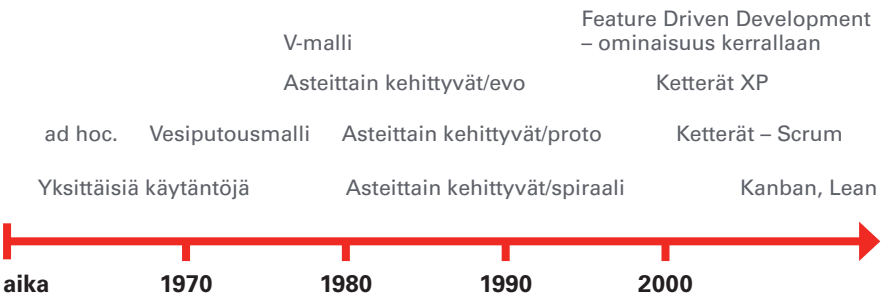
Ohjelmistotuotanto tai ohjelmistotekniikka, mitä termiä myös käytetään, on nuori ala, korkeintaan 50-60 vuotta vanha. Tuokin arvio on hieman liioittelua, sillä vaikka tietokoneet tulivat 1940-luvulla, vasta oikeastaan 1970-luvulta voidaan puhua ohjelmistotuotannosta. Verrattuna vanhoihin insinöörialoihin, kuten rakennustaitoon tai sillanrakennustaitoon, ohjelmistotuotanto ja siihen liittyvät projektit tai työtavat ovat kovin nuoria. Ohjelmistotuotanto eroaa perinteisistä insinöörialoista oleellisella tavalla: ohjelmistotuotannossa on vaikea eriyttää prototyypin tai pienoismallin tekemistä varsinaisen tuotteen tekemisestä. Jos rakennuksesta tehdään pienoismalli, jokainen sen nähdessään ymmärtää, että varsinainen rakennus pitää vielä rakentaa. Mutta jos maallikolle näytetään ohjelmiston tai tietojärjestelmän prototyyppi, hän voi pitää sitä jo varsinaisena tuotteena. Ohjelmistoista on vaikea nähdä, mikä niiden valmiusaste on.

Ohjelmistojen maine ei ole mairitteleva, niissä on totuttu virheellisiin tuotteisiin. *Tietojärjestelmien hankinta* -tutkimuksessa (2013) tarkasteltiin uusien tietojärjestelmäratkaisujen kehityksen ja vanhojen tietojärjestelmäratkaisujen ylläpidon sekä valmisohjelmistojen räätälöintiprojekteja. Tutkimuksessa selvitettiin sellaisia ohjelmistoprojekteja, joissa työmäärä edustaa merkittävää osaa koko projektin kustannuksista (Tietojärjestelmien hankinta Suomessa 2013). Raportin tulokset havainnollistavat hyvin ohjelmistotuotannon vaikeuksia, joista yksi merkittävimmistä on asioiden määrittely:

> Tilaajien ja toimittajien vastaukset olivat hyvin mielenkiintoisella tavalla ristissä. 80 % tilaajista kertoi, että he tekevät määrittelyn itse tai koko prosessi muuten mennään läpi heidän ohjauksessaan. Tilaajien mielestä vain 12 % määrittelyistä tehdään toimittajan ohjauksessa. 8 % määrittelyistä tilaajat teettävät ulkopuolisilla tahoilla ennen varsinaisen toimittajan valintaa. Toimittajat päinvastoin kokivat, että 50 % tietojärjestelmähankkeiden määrittelyprosesseista kulki heidän ohjaksissaan. Heidän mielestään vain 45 % prosesseista meni niin, että tilaaja oli tehnyt määrittelyn jo ennen toimittajan valintaa tai se tehtiin tilaajan ohjauksessa toimittajan avustuksella. On hyvin erikoista, että tilaajien ja toimittajien mielipiteet määrittelyprosessin tavallisesta kulusta eroavat näin selkeästi toisistaan. Todennäköisesti totuus löytyy jostain välimaastosta, kyseinen ristiriita johtuu huonosta kommunikaatiosta osapuolien välillä tai varsi-

naisen määrittelyprosessin tulkintaeroista. (Tietojärjestelmien hankinta Suomessa 2013, 10–11)

Ohjelmistojen tekemiseen on etsitty ratkaisuja erilaisista prosessimalleista ja työtavoista, joita on nähty jo useita alan lyhyen historian aikana.




Kuvio 1. Ohjelmistokehityksen työtapojen historiaa

Vanhin näistä on ns. vesiputousmalli, jota on paljon käytetty ohjelmistoalalla, vaikka sen puutteet on tiedostettu (kuvio 1). Vesiputousmalli on vaiheittain lineaarisesti etenevä prosessi, joka usein piirretään vesiputouksen lailla eteväksi vaiheesta toiseen. Malli on käytännössä vastaava kuin järjestelmäsuunnittelun malli muissa insinööritieteissä. Vesiputousmallin ja eräiden muidenkin perusteellista määrittelyä painottavien prosessimallien kritiikki sai ilmiasunsa vuonna 2001 ns. *Agile Manifestossa*. Tuona vuonna 17 ohjelmistoalan ammattilaista kokoontui Utahissa, allekirjoitti manifestin ja otti käyttöön termin *agile* eli ketterä. Olennaista ketterille menetelmille oli omaksua vanhoista malleista hyväksi koettu inkrementaalinen lähestymistapa tuotteen kehittämiseen. Inkrementaalisuus tarkoittaa ohjelmistokehityksessä pienten osakokonaisuuksien toteuttamista kerrallaan. Kaikki ketterät menetelmät korostavat muutosten hyväksymistä ja nopeita inkrementtejä.

Monista ketteristä menetelmistä tänä päivänä suosituimmat ovat Scrum ja eräät konkreettiset työtavat XP-menetelmästä (*Extreme Programming*). Kuitenkin ohjelmistoalalla käytetään edelleen myös muita perinteisempiä malleja, mm. vesiputousmallia, koska ei ole yhtä ainoaa oikeaa prosessimallia vaan käytetty malli riippuu projektista. Tällä vuosikymmenellä ohjelmistoalalla puhaltaa jo uusia tuulia: alalla puhutaan massatuotannossa käytetyistä Kanban- ja Lean-työtavoista. Lean on tuotannonoh-

jauksen ajattelumalli, joka perustuu turhan työn välttämiseen ja joka on peräisin autoteollisuudesta. Kanban taas on Leanin periaatteita noudattava yksinkertainen toteutus ohjauksesta.

# Ammattitaidon kasvu innovaatioprojekteissa

Ohjelmistotuotantoprojektit ovat tärkeä osa tietotekniikan alan opiskelijoiden kasvamisessa ohjelmistotekniikan ammattilaisiksi. Ohjelmistotuotantoprojektien määrittely-, suunnittelu- ja toteutustehtävillä ratkotaan ongelmia, jotka ovat tyypillisiä niin sanottuja huonosti määriteltyjä ongelmia. Ongelmien määrittelyt ovat usein epätäydellisiä, eivätkä asiakkaat ja kehittäjät tiedä tarkkaan, mitä halutaan lopputulokseksi. Huonosti määritellyille ongelmille ei ole vain yhtä ainoata oikeaa ratkaisua tai niille ei ole ratkaisua ollenkaan. Tällaisiin ongelmanratkaisutehtäviin liittyviä osaamisia on yritetty ymmärtää tutkimalla kehittäjien kognitiivisia prosesseja. Koska suunnittelu ja ohjelmointi tehdään tiimeissä ja yhteistyössä, pelkästään yksittäisen kehittäjän työskentelyä tutkivia lähestymistapoja on kritisoitu riittämättömiksi (Sonnentag ym. 2006).

Metropolia Ammattikorkeakoulun tietotekniikan koulutusohjelmassa eri kursseihin liittyen opiskelija tekee opintojen kuluessa monia pieniä projekteja. Koska projektit ovat usein vain osa kurssia, täytyy projektien kuitenkin olla melko pieniä ja usein vain kahden tai korkeintaan kolmen opiskelijan yhteisiä. Lisäksi nämä projektit keskittyvät usein kurssien tavoin johonkin ohjelmistotuotannon osa-alueeseen. Ohjelmistotuotantoprojektissa, joka on kestoltaan ja laajuudeltaan tarpeeksi iso, opiskelija vasta riittävässä määrin törmää edellä puhuttuihin huonosti määriteltyihin ongelmiin.

Ohjelmistotuotannon asiantuntijuutta on usein kuvattu karttuvien kokemusvuosien perusteella, vaikka kokemusvuodet eivät suoraan kerro huippuasiantuntijuuden kehittymisestä. Sonnentag ym. (Sonnentag ym. 2006) yrittävät etsiä eroja ohjelmistosuunnittelun asiantuntijan ja ei-asiantuntijan välillä. He jakavat asiantuntijuuden viiteen eri osa-alueeseen: vaatimusten analysointi ja ohjelmiston suunnittelu, ohjelman ymmärtäminen ja ohjelmointi, testaus- ja virheenjäljitystaito sekä kommunikointi ja yhteistyö. Näillä osa-alueilla Sonnentag ym. yrittävät erottaa, miten asiantuntijoiden toiminta eroaa ei-asiantuntijoiden työtavoista ja -malleista. Näillä viidellä osa-alueella pyritään käsitteellistämään asiantuntijuutta eikä vain kuvaamaan työssä kulutettua aikaa.

Kun yritämme kehittää opiskelijoiden osaamista, oppimistavoitteisiin liitetään niitä odotuksia, jotka edistävät asiantuntijuuden kehittymistä. In-

novaatioprojektit ovat koko Metropolia Ammattikorkeakoulun tasolla määriteltyjä opiskelijaprojekteja, jotka tulivat opetussuunnitelmiin kolmannen vuoden opintoina. Ensimmäisen kerran innovaatioprojekteja toteutettiin vuonna 2010. Näiden innovaatioprojektien tavoitteissa kerrotaan, että projektin jälkeen opiskelija osaa

- soveltaa projekti- ja verkostotyöskentelyn perusteita ja osaamistaan alueellisessa, valtakunnallisessa tai kansainvälisessä kehittämistyössä
- hyödyntää omaa asiantuntijaosaamistaan ja tuoda sen moniasiantuntijuuteen perustuvaan toimintaan
- luoda yhteistoiminnallista neuvottelukulttuuria muiden toimijoiden kanssa
- käyttää ongelmaratkaisu-, yhteistyö- ja viestintätaitojaan yhteisöllisessä kehittämisprosessissa ja päätöksenteossa sekä
- kehittää muiden toimijoiden kanssa käytännöllisiä, luovia ja innovatiivisia ratkaisuja, toimintatapoja tai palveluja, joilla vastataan metropolialueen monimuotoisiin tarpeisiin.

Tietotekniikan koulutusohjelman ohjelmistotekniikan suuntautumisessa tärkeä osa opiskelijoiden kehittymistä ohjelmistoammattilaisiksi on ohjelmistoprojekteissa toimiminen ja nimenomaan isohkon ohjelmiston suunnittelu, toteutus ja testaus. Niinpä Metropolian tietotekniikan koulutusohjelmassa ohjelmistotekniikan suuntautumisessa on toteutettu puoli vuotta kestäviä 4–6 hengen ohjelmistotuotantoprojekteja. Nämä ovat ohjelmistotekniikassa innovaatioprojekteja eli ns. *capstone*-projekteja, joissa taidot näytetään. Opiskelijat muodostavat projektiryhmän, joka suunnittelee ja toteuttaa sovelluksen itsenäisesti.

Metropolian innovaatioprojektien tavoitteita täydentävät ohjelmistotuotantoprojektin tavoitteet. Opintojakson tavoitteina on oppia

- työelämässä käytettyä Scrum-menetelmää
- asiakkaan kanssa toimimista
- työmäärien arviointia
- projektityöskentelyä ja ryhmätyöskentelytaitoja
- SM:n[1] ja PO:n[2] roolin toimintaa sekä
- jatkuvaa integrointia ohjelmistotuotannossa.

---

1 *Scrum Master*

2 *Product Owner*

Ohjelmistotuotantoprojektit ovat koulutusohjelman tapa tarjota opiskelijoille isohkon projektin läpivientiä oppimisprojektina, tuettuna ja työelämässä käytettyjä työtapoja opiskellen. Ohjelmistotuotantoprojektin tavoitteissa paneudutaan nimenomaan sen oman alan asiantuntijuuden kehittämiseen, jota luvun alun asiantuntijuuden osa-alueista puhuttaessa pohdittiin.

# Scrum – ketterä prosessi

Ohjelmistotuotantoprosesseissa on siirrytty viimeisen kymmenen vuoden aikana ns. ketteriin menetelmiin. Yksi suosituksi tullut prosessimalli on Scrum. Se perustuu ajattelutapaan, jossa tunnustetaan, että ohjelmistoprojektia on mahdotonta määritellä heti alussa. Usein projektit ovat luonteeltaan uutta tuottavia tuotekehitysprojekteja, joissa asiakaskaan ei vielä tarkasti tiedä, mitä pitäisi tehdä. Menetelmän nimi tulee rugby-pallopelistä. Pelin aloitusryhmityksen tiiviistä ryhmästä käytetään nimitystä scrum tai scrummage. Nimi on jäänyt käyttöön johtuen menetelmän yhtäläisyyksistä rugby-joukkueeseen ja sen toimintaan; molemmat ovat erilaisiin tilanteisiin sopeutuvaisia, nopeita ja itseohjautuvia.

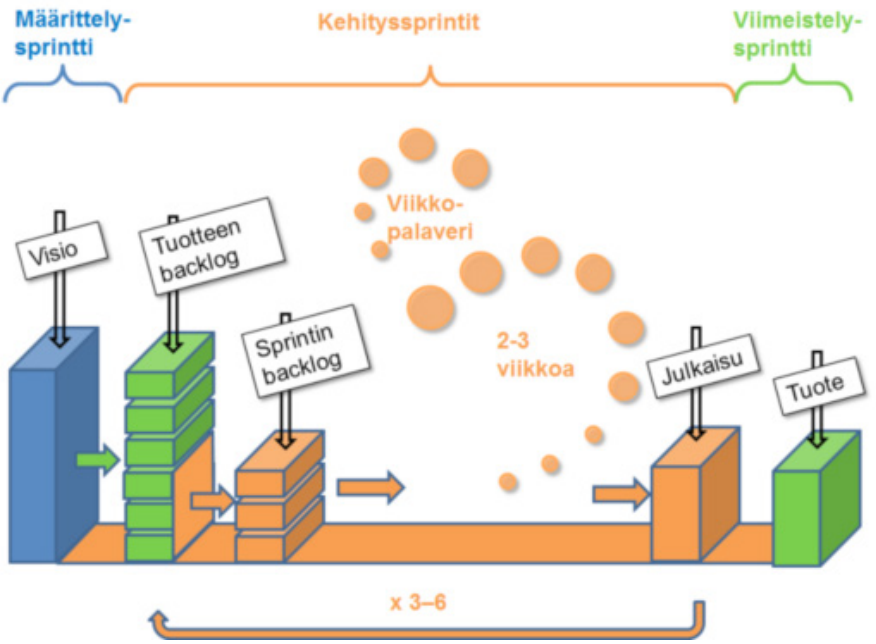
Scrum-projektissa tuotteenomistaja eli asiakas määrittelee tuotteen ominaisuuslistan, jossa asiakas tuo esille, mitä haluaa tuotteelta tai sovellukselta. Tämän listan asiakas järjestää tärkeysjärjestykseen. Listalla olevien ominaisuuksien ei tarvitse olla mitenkään samansuuruisia tai yhteismitallisia, eikä niitä kaikkia tarvitse heti alkuun määritellä tarkasti. Listalta projektiryhmä poimii tärkeysjärjestyksessä ne toiminnallisuudet, jotka seuraavaksi aiotaan suunnitella ja toteuttaa. Projektiryhmä sitoutuu tekemään valitsemansa ominaisuudet. Tuotteen ominaisuuslistassa voisi olla esimerkiksi seuraava ominaisuus:

> *”Ostajana haluan lisätä konserttilipun ostoskoriin.”*

Esimerkistä käy ilmi kuka tätä ominaisuutta haluaa (= ostaja). Usein tuotteilla on erilaisia käyttäjiä ja heidän haluamansa ominaisuudet poikkeavat toisistaan. Esimerkissä on kyse verkkokaupan tapaisesta sovelluksesta. Sovelluksen halutaan toimivan niin, että tuotteessa on ostoskori, johon voi lisätä valitsemiaan lippuja. Näitä ominaisuuksia tarkennetaan ja näistä muodostuu yhden lyhyen työrupeaman eli sprintin toiminnallisuuslista, jonka perusteella projektiryhmä tarkentaa ne tehtävät, jotka sprintissä täytyy saada tehdyksi. Tehtävistä arvioidaan niihin kuluva työaika, ja työtä tehtäessä ajankäyttöä seurataan. Sprintit ovat olennainen osa Scrumia. Projektin alussa valitaan sprintille pituus, esimerkiksi neljä viikkoa, ja tämä sprintin pituus säilyy samana koko projektin ajan eikä se jousta työn etenemisen mukana.

Scrumin ketteryys tässä hieman kangistuu, mutta tällaiselle työskentelytavalle on perusteita. Kun sprintin koko on sama koko ajan, projektitiimi

oppii vähitellen, kuinka paljon työtä voi ottaa vastaan tehtäväksi. Toinen tärkeä peruste on työn rauhoittaminen sprintin ajaksi. Kun sprint suunnitellaan ja päätetään, mitkä ominaisuudet seuraavaan sprintiin otetaan, tarkoitus on, että uusia ominaisuuksia ei sprintin aikana tule. Näin projektitiimi saa sprintin ajaksi työrauhan. Joskus käy niin, että asiakas vaatii sprintin muutosta, mutta silloin Scrumin periaatteiden mukaan jotain täytyy ottaa sprintistä pois. Jos jotain ominaisuutta ei ole vielä aloitettu, se voidaan silloin vaihtaa asiakkaan haluamaan.




Kuvio 2. Sovellettu Scrum

Kuviossa 2. on kuvattu ohjelmistotuotantoprojekteissa sovellettu tapa käyttää Scrumia. Työelämässä Scrum-työtapa on nykyään laajalle levinnyt menetelmä tehdä ohjelmistotyötä (Deemer ym. 2012). Projektit noudattavat Scrumin periaatteita, mutta koska opiskelijat eivät ole kokopäivätyössä projekteissa, muutamia pieniä räätälöintejä on tehty. Lisäksi opiskelijaprojekteja varten on ohjelmistotuotantoprojektissa kaksi lyhyttä sprinttiä: määrittelysprintti ja viimeistelysprinttti. Nämä ovat kestoltaan vain kaksi viikkoa. Määrittelysprintissä opiskelijat tuottavat sovelluksesta ns. toiminnallisen määrittelyn ja viimeistelysprintissä opiskelijat tuottavat projek-

tin yhteenvetoraportin. Tarkemmin käytettyä Scrum-prosessia selostetaan seuraavissa luvuissa.

# Projektitilat tiimille

Perinteiset luokkatilat olivat ainoat käytettävissä olevat tilat, kun ohjelmistotuotantoprojektit aloitettiin. Projektiryhmät joutuivat työskentelemään tavallisissa tietokoneluokissa, kunnes saatiin rakennettua ensimmäinen projektitila, jossa opiskelijat istuvat yhteisen pöydän ääressä ja jossa heidän käytössään on kannettavat tietokoneet. Samassa yhteydessä otettiin käyttöön Scrum-menetelmä.

Kuviossa 3. opiskelijat pelaavat Scrum-peliä, jossa tehtävien vaatimaa työaikaa arvioidaan. Kukin projektin jäsen esittää oman arvionsa työmäärästä, kunnes perustelujen ja keskustelujen kautta päädytään yhteisymmärrykseen työmääräarviosta.

Kuvio 3. Scrum-arviointipeliä pelaamassa (kuva: Erja Nikunen 2012)

Työtilan vaihtuminen oikeaksi projektitilaksi muutti suuresti työtapaa. Keskustelu oli helppoa, jokaiselta tietokoneelta voitiin heijastaa kuvaa seinälle, ja valkotauluille pystyttiin hahmottelemaan ratkaisuja. Myöhemmin projektiluokkiin hankittiin vielä näytöt, joissa näkyy viimeisimmän koonnin tila, esimerkiksi:

- Onko käännös onnistunut?
- Ovatko testit menneet läpi?

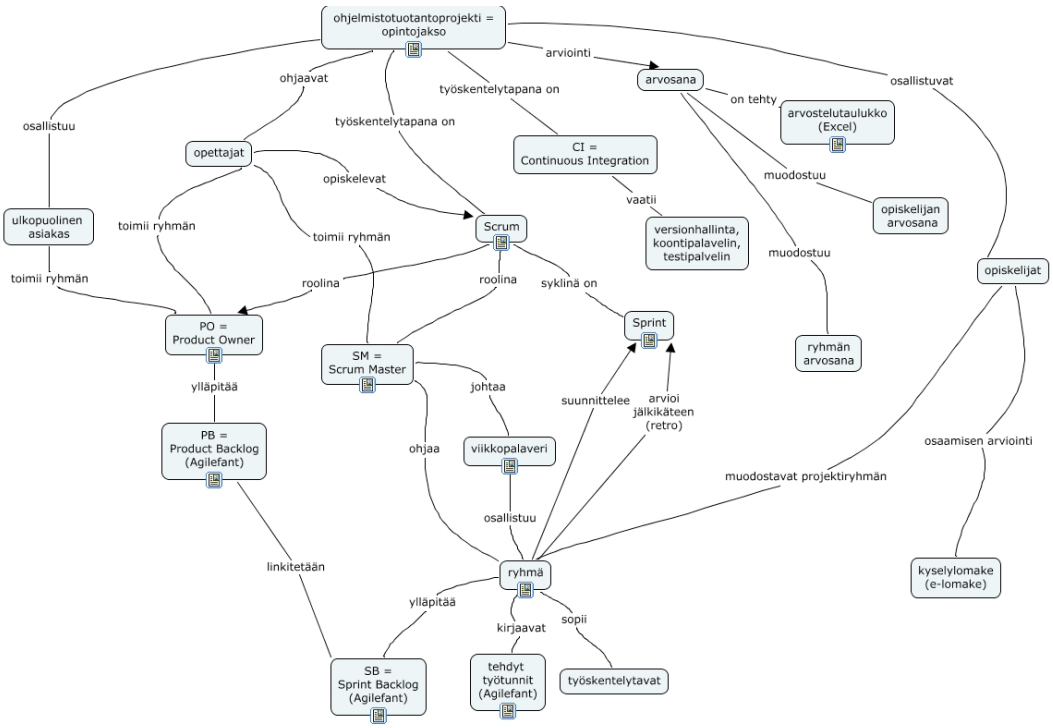
Koonniksi kutsutaan kehitystyössä prosessia, joka kokoaa ohjelmiston komponentit yhteen, kääntää ja mahdollisesti linkittää osat ja suorittaa kaikki automaattitestit.

Kaikki projekteissa tarvittavat ohjelmistotyökalut on tehty niin sanotuilla *open source* -työkaluilla, ja näin ollen työkalujen osto- ja lisenssimaksut säästyvät. Open source -ohjelmistoilla tarkoitetaan ohjelmia, joita kuka tahansa voi käyttää, tutkia ja muokata. Lisäksi olennaisena osana on muutosten jakelu edelleen eteenpäin, eli muuttuneiden lähdekoodien pitää olla kaikkien saatavilla. Ympäristön toteutus on tehty opinnäytetöinä (Lukkarinen 2011; Kasari 2012) ja harjoitteluna. Tietokoneiden ja verkon pystytykseen on saatu apua Metropolian tietohallinnolta.

# Kohti vakiintuneita työtapoja

Vuosien mittaan työskentelytavat ovat tietotekniikan koulutusohjelmassa erilaisten kokeilujen kautta vakiintuneet. Kuviossa 2. näkyi, mitä sprinttejä on käytössä olevassa mallissa: määrittelysprintti (kaksi viikkoa), viimeistelysprintti (kaksi viikkoa) ja kehityssprintit. Kehitysprinttejä on projektissa käytössä olevan sprintin pituudesta riippuen 3–6. Tällä tavalla opiskelijat aluksi tutustuvat ongelmaan ja kuvaavat asiakkaan haluaman ratkaisun pääpiirteissään. Sen jälkeen kehitysprinteissä määritellään tarkemmin sprintti kerrallaan, mitä tehdään. Kun kehitysprintin kesto on sovittu, se pysyy samana koko ajan. Opiskelijaryhmän täytyy oppia arvioimaan suunnitelmaansa ja työmääriä paremmin ja paremmin.

Opintojaksoon osallistuneet opettajat kävivät läpi Tieturin Ketterän projektin läpivienti -nimisen kolmipäiväisen koulutuksen Scrum-projekteista. Tästä eteenpäin projekteissa käytetyt työskentelytavat ja roolit ovat kirkastuneet ja toimintaa voi kuvata seuraavalla käsitekartalla.

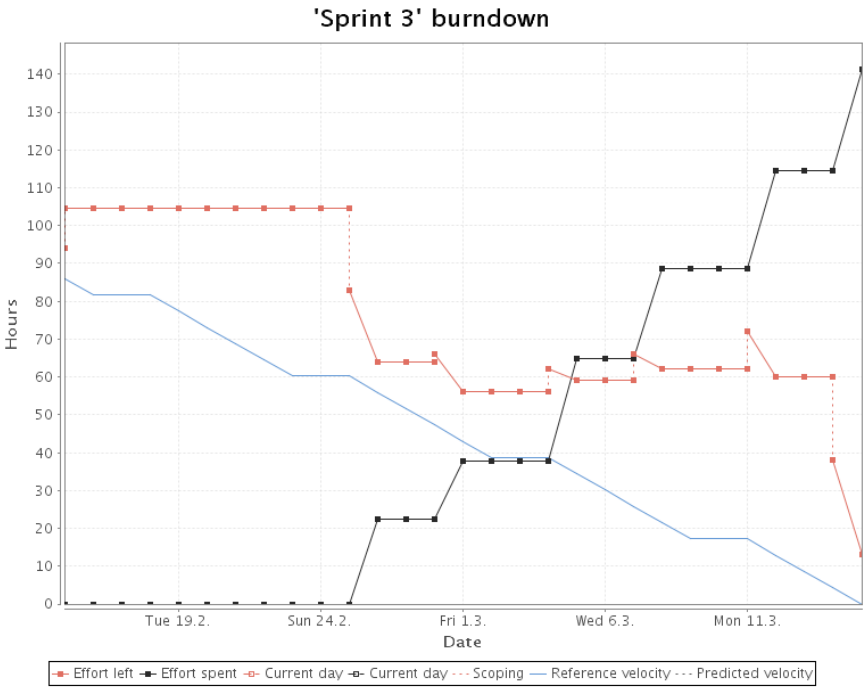



Kuvio 4. Käsitekartta projektissa käytetyistä menetelmistä ja työtavoista

Scrum-tyylisesti joko yksi opettajista tai ulkopuolinen asiakas toimii tuotteenomistajana eli *Product Owner* -roolissa. Hänen tehtävänään on kertoa, mitä asiakas haluaa, ja hän päivittää tuotteen ominaisuuslistaa. Yksi opettajista toimii *Scrum Master* -roolissa, ja hänen tehtävänään on pitää Scrum-palaverit. Koska opiskelijat eivät tee työtään joka päivä vaan työskentelevät noin 20 tuntia viikossa ja lukujärjestykseen merkittynä tunteja on vain maanantaisin ja torstaisin, Scrumin päiväpalaverin korvaa viikkopalaveri. Viikkopalaverissa katsotaan, mitä kukin on tehnyt edellisellä viikolla ja mitä aikoo tehdä tulevalla viikolla. Lisäksi sprinttien alussa on aina suunnittelupalaveri, jossa työskentely jaetaan tehtäviin, joille merkitään arvioidut työtunnit ja vastuuhenkilöt. Sprintin lopussa on vastaavasti retrospektiivinen kokous, jossa tarkastellaan, missä onnistuttiin ja mitä voidaan parantaa.

Edellä kuvattu työskentelytapa toteutetaan prosessina, jossa käytetään tärkeänä reaaliaikaisena apuvälineenä Agilefant-projektinhallinnan työkalua (Vähäniitty ym. 2010). Agilefant on web-selaimessa toimiva open source -työkalu, jota kehitetään MIT-lisenssillä ja josta vastaa SoberIT:n Software Process Research Group Aalto-yliopistossa. Se on suunniteltu erityisesti Scrum-projektien tueksi. Ryhmä kirjaa Agilefantiin asiakkaan määrittelemän ominaisuuslistan. Työkalu tukee sprint-ajattelua, joten ominaisuudet saadaan helposti siirretyksi sprinttien toiminnallisuuslistoiksi. Projektiryhmä pilkkoo toiminnallisuudet sprinttien tehtäviksi (task). Kukin projektiryhmän jäsen näkee Agilefantista senhetkisen tilanteen: mitkä toiminnallisuudet tai tehtävät on toteutettu ja mitkä asiat ovat tekeillä ja kuka niitä tekee? Kun opiskelija saa valmiiksi tekeillä olevan tehtävän, hän valitsee listasta seuraavan tekemättömän tehtävän. Tehtävää tehtäessä ja sen valmistuttua Agilefantiin kirjataan toteutunut työaika. Näin voidaan seurata arvioitujen ja toteutuneiden työaikojen suhteita ja nähdään reaaliajassa sprintin arvioitu jäljellä oleva työmäärä. Koska asiakkaalla on pääsy Agilefantiin, hän voi seurata projektin etenemistä reaaliajassa.

Agilefant tarjoaa ympäristön, jossa opiskelijat suunnittelevat yhteisöllisesti omat tehtävänsä, kirjaavat työn edistymisen ja voivat seurata toistensa töiden etenemisiä. Kuviossa 5. sprintin edistymiskäyrässä eli burndown-kuvassa musta murtoviiva kuvaa projektiryhmän tekemän työn lisääntymistä, punainen viiva arvioitua jäljellä olevaa työmäärää ja sininen viiva optimaalisesti etenevää projektia. Kuviosta näkee, että projektiryhmä ei alkuun ole tehnyt mitään, mutta sitten tehty työ on hypähdyksittäin kasvanut. Kuitenkin punaisesta viivasta näkee, että koko ajan on oltu optimaalisen projektinkulun yläpuolella eli on arvioitu jäljellä olevan enemmän




Kuvio 5. Burndown-kuva eräästä sprintistä

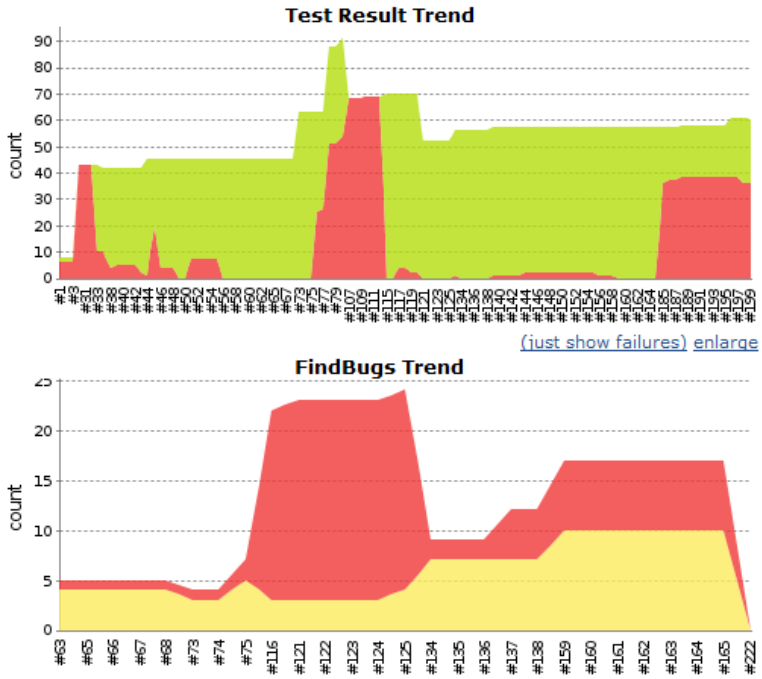
työtä kuin optimaalisesti etenevässä projektissa pitäisi. Lopussa näkyy, että tekemätöntä työtä jäi jäljelle. Muutama ylöspäin hyppäys punaisessa viivassa kertoo, että työn arvioinneissa on tapahtunut uudelleenarviointia: työtä onkin enemmän kuin oli alun perin arvioitu.

# Laadunvarmistusta

Ketterien menetelmien yhteydessä otetaan yleensä käyttöön työvälineitä, joilla varmistetaan tuotteen laatua. Jatkuva integrointi pyrkii automatisoimaan tuotantoprosessia kaikilta niiltä osin kuin on mahdollista. Rutiinimaisten ja työläiden vaiheiden automatisoinnilla vähennetään virheitä ja nopeutetaan työrytmiä. Ongelmat saadaan projektiryhmän tietoon nopeasti ja ongelmiin voidaan näin reagoida heti (Pilone ym. 2007).

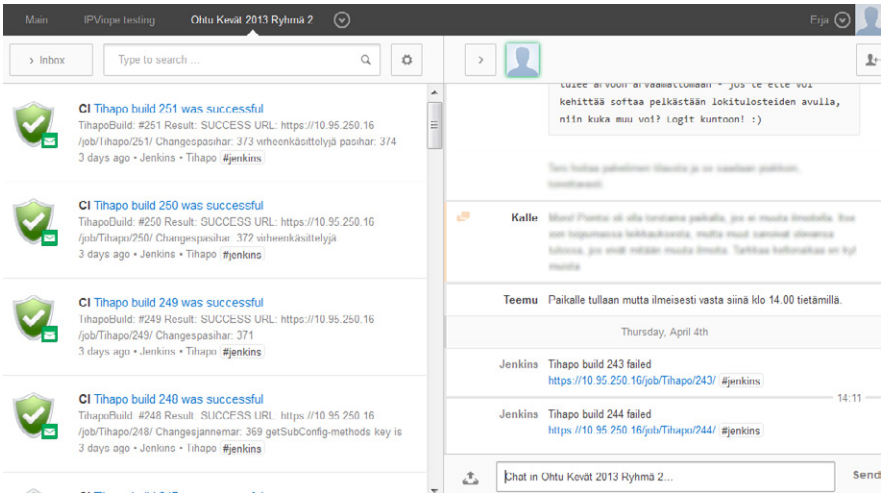
Olennainen osa ohjelmistotuotantoprojektien työskentely-ympäristöä ovat automaattiset koonti- ja testaustyökalut. Java-sovellusten kehittämistä varten tietotekniikan koulutusohjelmassa otettiin käyttöön ensimmäistä kertaa automaattinen kehitysympäristö. Järjestelmästä saadaan tietoa, miten kehitettävän sovelluksen viimeinen koonti on mennyt ja kuinka testit ovat sujuneet. Automaattisella koonnilla ohjelmakehityksessä tarkoitetaan, että aina kun joku kehittäjä on tehnyt muutoksen ohjelmakoodiin versionhallinnassa, suoritetaan ohjelmiston kääntäminen ja kokoaminen. Samassa yhteydessä ajetaan myös kaikki testit, joiden avulla voidaan varmistua sovelluksen toimivuudesta. Tällainen työskentelytapa on olennainen osa ohjelmiston laadunvarmistusta: jokaisen muutoksen jälkeen pyritään varmistamaan, että tehty työ on laadukasta. Järjestelmää on laajennettu toimimaan myös muiden kuin Java-projektien kanssa.

Automaattinen koontiympäristö tarjoaa välineen, joka mahdollistaa sen, että kukin projektin jäsen voi seurata myös reaalituotteen valmiusasteen etenemistä. Siinä missä Agilefant on suunnittelun ja raportoinnin työkalu, automaattinen koontiympäristö eli jatkuvan integroinnin Jenkins-palvelin mahdollistaa lopputuotteen testaamisen kattavuuden ja valmiusasteen seuraamisen (Jenkins 2013). Automatiikan ansiosta projektiryhmä voi seurata tilannetta jatkuvasti (24/7) ja hälytykset poikkeamista voidaan raportoida välittömästi projektin jäsenille esimerkiksi sähköpostilla tai tekstiviestein.




Kuvio 6. Jenkins CI[3]-palvelimen aikasarjaa koonneista

Kuviosta 6. näkee, kuinka monta läpimennyttä testiä on ollut eri vaiheissa (vihreät) ja kuinka monta epäonnistunutta testiä (punaiset). Alempi osa kuviosta 6. kertoo, kuinka paljon virheitä staattisen analyysin työkalu FindBugs on löytänyt projektin koodista.




Kuvio 7. Flowdock-tiimiviestintäsovelluksen näyttö

---

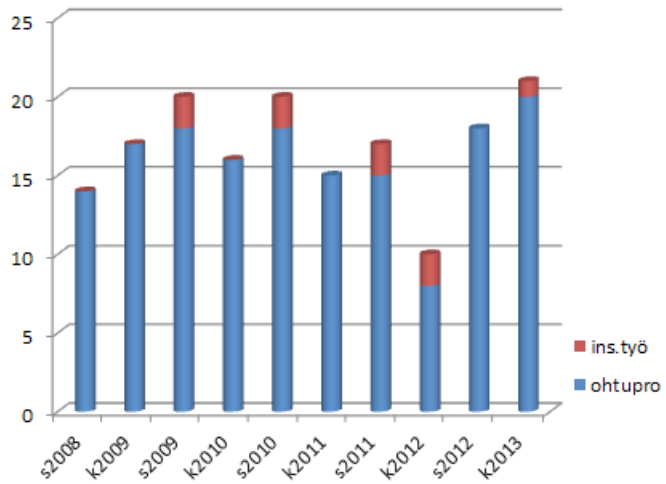
3 Continuous Integration

Projektissa on kokeiltu myös Flowdock-työkalua. Flowdock on web-pohjainen tiimiviestintätyökalu, joka pohjautuu ryhmäkeskusteluun ja integroituu kaikkiin tiimin käyttämiin työkaluihin, kuten sähköpostiin ja versionhallintatyökaluihin (Flowdock 2012). Kuviossa 7. Flowdock-työkalu on yhdistetty jatkuvan integroinnin palvelimeen, jolloin koontien tulokset näkyvät tietovirrassa.

# Onnistumisen kokemuksia ja asiakasyhteistyötä

Projektin arviointi perustuu sekä yksilösuoritukseen että ryhmän toiminnan arviointiin. Projektin lopuksi opiskelijat arvioivat projektia, omaa suoritustaan projektiryhmässä, projektiryhmän suorituksia, tehtävien vastuunjakoa ja itse opintojakson järjestelyjä. Projekteissa edellytetään opiskelijoiden läsnäoloa. Silloin tällöin on mietitty, voisiko tehtyjen tuntien määrä vaikuttaa suoraviivaisesti arvosanaan tai jopa opintopisteiden kertymään. Tällaisiin menettelytapoihin ei ole kuitenkaan vielä menty. Toistaiseksi jokainen on saanut saman verran opintopisteitä, mutta arvosana on määräytynyt sekä ryhmän että yksilöpanoksen perusteella.

Vuosien aikana opintojaksossa on ollut mukana yhteensä 159 opiskelijaa ja 8 opettajaa eri rooleissa (kuvio 8.). Projekteihin liittyviä insinöörityötä on tehty yhdeksän. Opiskelijat eivät ole keskeyttäneet projektia. Yksi hyvän onnistumisen takaaja on ollut ryhmien kokoamistapa. Projektiryhmässä tarvitaan erilaista osaamista: johtamistaitoa, kommunikointitaitoja, vastuunottokykyä, vahvaa ohjelmointiosaamista, kirjoittamisosaamista jne. Projektin ongelma-alueeseen perustuen varmistetaan aiemmin suoritettu-




Kuvio 8. Opiskelijamäärät (ins.työ = insinöörityöiden lukumäärä, ohtupro = ohjelmistotuotantoprojektiin osallistuneiden lukumäärä)

jen kurssien avulla teknisen osaamisen riittävyys. Ryhmien muodostamisen perusteena on ollut, että jokaiseen ryhmään kootaan erilaisia osaajia. Tapa on myös muualla osoittautunut menestyksellisesti. Lisäksi on pyritty myös rikkomaan vakiintuneiksi muodostuneet työskentelyryhmät.

Aluksi projekteja tehtiin sisäisistä aiheista, sitten mukaan tuli kone- ja tuotantotekniikan aiheita, ja nyttemmin on projekteissa ollut ulkopuolisia asiakkaita yritysmaailmasta. Kone- ja tuotantotekniikan koulutusohjelman kanssa on tehty yrityksen tuoteprosessien mallintamissovellus ja materiaalien testauksen raportointisovellus. Ulkopuolisen asiakkaan kanssa on tehty sovellus, joka liittyy sähkökomponenttien tunnistamiseen sähköpiirustuksista. Suomalaisen pelifirman kanssa on tehty Kalevala-aiheinen pelisovellus, jossa pelin hahmot tehtiin Intiassa ja varsinainen pelimoottori ohjelmistotuotantoprojektissa. Accenturen kanssa on tehty sovellus oppilaitosten kanssa tehtävää yhteistyötä varten. Rakennustekniikan koulutusohjelman ja rakennusalan yrityksen kanssa on tehty rakennustarkastuksia varten mobiilisovellus.

Projekteissa on kokeiltu ohjelmiston laadunvarmistusmenetelmänä myös ulkopuolisen laaduntarkkailijan suorittamaa arviointia. Neljästä ryhmästä valittiin kaksi, joiden prosessia ja menettelytapoja arvioitiin. Arvioijalle toimitettiin kuvaukset ohjelmistotuotannon käytännöistä etukäteen ja arviointisessioissa projektiryhmän jäsenet vastasivat arvioijan kysymyksiin. Arvioinnista tehdään raportti projektiryhmän käyttöön.

## Opiskelijoiden itsearviointia

Opettajien ja koulutusohjelman kokemusten lisäksi opiskelijoiden omat kokemukset tämäntyyppisestä projektista ovat ensiarvoisen tärkeitä. Jokaisen ohjelmistotuotantoprojektin lopussa opiskelijat täyttävät itsearvioinnin, jonka yhteydessä kysytään mm. ”Mitä asioita opit tämän projektin aikana?”

> *”Yleisestikin ohjelmistotuotantoprojektin vaiheet, toteutusosiot ja toimenpiteet. Myös projektinhallintaa Agilefantilla ja Jenkinsillä. Tarkemmin sanottuna JUnit-testaus, testitapausten kirjoittaminen hyväksymistestaukseen, koko ohjelmiston jokaisen ominaisuuden testaus ja virheiden raportointi sekä parannusehdotukset ryhmämme jäsenille. Lisäksi asiakaspalaverit, dokumentointi ja syötevirheiden validointi.”*

*”Projektin aikana opin huomaamaan mahdolliset sudenkuopat ja ymmärtämään paremmin projektinhallinnan tärkeyden ohjelmistotuotannossa. Ohjelmiston kehityksen myötä huomasin myös määrittelyn jälkeisen alustavan arkkitehtuurisuunnittelun tärkeyden. Scrummaustaitoja sain myös hiottua. Projektin lopussa opin hyvän ryhmähengen tärkeyden ja miten paljon mukavampi on tehdä töitä mukavassa työympäristössä.”*

*”Uudet koodikielet, tärkeys kommunikoida asiakkaan kanssa projektin kehityssuunnasta sekä lisää ryhmätyöstä ja sen tärkeydestä.”*

Itsearviointien perusteella voi sanoa, että opiskelijat tunsivat käytännön taitojensa syventyneen lähes kaikissa Johdannossa esitellyissä asiantuntijuuden osa-alueissa. Samoin opiskelijoiden palaute kertoo innovaatio- ja ohjelmistotuotantoprojektien oppimistavoitteiden toteutumisesta. Lisäksi opiskelijat ovat projektissa nähneet käytännössä sellaisten asioiden tärkeyden, jotka eivät ole aiemmilla kursseilla välttämättä vielä selkiytyneet.

Projektin kehittämisestä opiskelijoilta kysyttiin: ”Miten ohjelmistotuotantoprojektia pitäisi mielestäsi kehittää (työmenetelmät, työkalut, ohjaus, tms.)?”

*”Määrittely voisi olla lyhyempi ja aluksi tulisi vain kriittisimmät käyttäjävaatimukset. Suunnitteluun voisi käyttää reilusti enemmän aikaa, jotta saataisiin joustava ja kaikkien ymmärtämä pohja ohjelmalle. Suunnittelussa kannattaisi käydä ryhmässä läpi reilusti mahdollisia toteutusvaihtoehtoja ja lopuksi valita kannattavin. Ohjauksen tulisi olla tiukempaa ja tarttua nopeammin ryhmän virheelliseen toimintaan.”*

*“Olisin kaivannut ohjausta/apua Spring-kehykseen liittyen. En ole lainkaan varma, että kaikki ratkaisuni ovat järkeviä. Osat asiat olisi voitu varmaankin tehdä helpommin ja muutama lisäominaisuus olisi voitu toteuttaa enemmän, kunhan olisi saatu lisäopetusta/ohjeita. Yksikkötestaukset jäivät esim. kokonaan toteuttamatta jne. Mielestäni Agilefant toimi hyvin projektin tehtävien jakamisessa ja tilanteen seuraamisessa.”*

*”Tämä toimii hyvin.”*

Edellä olevien vastausten perusteella siirryttiin nykyiseen käytäntöön, jossa alussa on vain lyhyt määrittelysprintti ja vasta sen jälkeen tulevat ke-

hityssprintit. Muutamassa projektissa on pystytty asiakkaan puolelta saamaan ohjausta ja koulutusta uusien tekniikoiden ja työvälineiden käytössä.

Reflektointi projektin ja itse asiassa jokaisen sprintin lopussa on tärkeä osa oppimista. Opiskelijat arvioivat tuloksia omatoimisesti ja myös asiakkaan kanssa yhdessä. Uuden oppimista niin ohjelmiston ymmärtämisessä, ohjelmoinnissa kuin testauksessakin tapahtuu projektin edetessä. Keskeistä on oivaltaa, mitä asioita olisi projektin aikana itse voinut tehdä paremmin.

> *"Muutamaan otteeseen aloitin ohjelmoinnin liian aikaisin, ja kesken toteutusta keksin/ymmärsin paremman ratkaisun ongelmaan, jolloin joutui palaamaan suunnittelupöydän ääreen."*

# Lopuksi

Eri kursseilla opitut asiat yhdistyvät innovaatioprojektissa työtavaksi, joka sitoo esimerkiksi testauksen, projektinhallinnan, versionhallinnan ja tietenkin ydinosaamisen eli ohjelmoinnin yhteen. Opiskelijoiden itsearvioinneissa korostuvat kommunikaation ja ryhmätyöskentelyn oppimiskokemukset. Testausosaamisessa on havaittu olevan puutteita ja tämän takia testaustaidon ja ohjelmien ymmärtämisen opettelua on viety aiempaa alempien vuosikurssien ohjelmointikursseille. Jo olio-ohjelmoinnin ensimmäisellä opintojaksolla opetellaan käyttämään valmiita testejä, minkä jälkeen opetellaan itse tekemään testejä. Aiemmin testausta varsinaisesti opeteltiin vasta ohjelmien testaus ja hallinta -opintojaksolla kolmantena vuonna.

Gotel ym. (2009) raportoivat monivuotista tutkimustaan, jossa he neljän vuoden ajan tutkivat opiskelijoiden työtapoja. Tutkimuksessa kehitettiin vuosi vuodelta työympäristöjä lähtien aluksi hyvin vaatimattomasta kehitysympäristöstä. Vasta viimeisenä vuonna keskityttiin version- ja konfiguraationhallintaan ja jatkuvaan integrointiin. Opiskelijat arvioivat hankalaksi omaksua koodin jatkuvan parantamisen ja integroinnin sekä regressiotestauksen. Sarma ym. (2008) kertovat raportissaan kokeellisesta tutkimuksesta, jossa Eclipseen tehtiin liitännäinen, jonka avulla selvitettiin version- ja konfiguraationhallintatyökalujen käyttöä projekteissa. Eclipse on paljon käytetty graafinen ohjelmankehitysympäristö.

Ohjelmistotuotannon osaamisia on tutkittu monissa muissakin tutkimuksissa, mutta useimmat keskittyvät jonkin pienen osa-alueen selvittämiseen. Ohjelmointiosaamista on tutkittu paljon (esim. Bornat ym. 2008), samoin kuin laajojen API-kirjastojen[4] käytön helppoutta tai erilaisten IDE-ympäristöjen[5] käytettävyyttä. API-kirjastot eli sovellusrajapinnat tarjoavat ohjelmoijalle suuren määrän jo valmiiksi toteutettuja ohjelmia. IDE-ympäristöt graafisina työskentely-ympäristöinä tarjoavat ohjelmointiin graafisen käyttöliittymän. Molempia on tutkittu helppokäyttöisyyden kannalta. Kokonaisuutta tai sitä, miten asiantuntijuus kehittyy, on tutkittu melko vähän. Ohjelmistotuotantoprojektien myötä saadaan tietoa siitä, mitkä asiat ovat jääneet liian vähäiselle harjoitukselle edeltävissä opinnoissa.

---

4 *Application Programming Interface*

5 *Integrated Development Environment*

Ohjelmistotuotantoprojektien yksi tärkeimpiä motivoivia tekijöitä ovat asiakkaiden mukanaolo ja heidän mukanaan tuomansa ongelmat ja aiheet, joihin projektiryhmänä yhdessä haetaan ratkaisua. Monialaisuus tulee projekteissa esille asiakkaiden kautta: projekteissa on muun muassa toteutettu peliä antikoagulaatiohoidon opettamiseen ja opiskelijarekisterin konvertointia kansalliseen Virta-tietovarantoon.

**Lähteet**


Bornat, R. – Dehnadi, S. – Simon 2008: Mental models, consistency and programming aptitude. S. 53–61 of: ACE '08: Proceedings of the tenth conference on Australasian computing education. Darlinghurst, Australia, Australia: Australian Computer Society, Inc.

Deemer, P. – Benefield, G. – Larman, G. – Vodde, B: 2012 The Scrum Primer. Saatavissa osoitteessa: <http://www.scrumprimer.com/>. Luettu 23.4.2013.

Ericsson, K. A. – Charness, N. – Feltovich, P. J. – Hoffman, R. R. (toim.) 2006: The Cambridge Handbook on Expertise and Expert Performance. Cambridge Univ. Press.

Flowdock-sovelluksen kotisivu. Saatavissa osoitteessa: <https://www. flowdock. com/help>. Luettu 23.4.2013.

Gotel, Olly – Kulkarni, Vidya – Phal, Des – Say, Moniphal – Scharff, Christelle, – Sunetnanta, Thanwadee. 2009: Evolving an Infrastrusture for Student Global Software Development Projects: Lessons for Industry. S. 117– 126. ISEC '09, Pune, India. ACM.

Jenkins, continuous integration. Projektin sivusto saatavissa osoitteessa: <http://jenkins-ci.org/>. Luettu 23.4.2013.

Kasari, Topi 2012: Jatkuvan integroinnin koontityökalut. Metropolia Ammattikorkeakoulu. Saatavissa osoitteessa: <http://urn.fi/URN:NBN:fi:amk-201204244969>.

Lukkarinen, Aleksi 2011: Iteratiivinen sovelluskehitys: Kehitysympäristön toteutus ja ylläpito. Metropolia Ammattikorkeakoulu. Saatavissa osoitteessa: <http://urn.fi/URN:NBN:fi:amk-201105269796>.

Pilone, D. – Miles, R. 2007: Head First Software Development. O'Reilly Media, Inc.

Sarma, A. – Redmiles, D. – van der Hoek, A. 2008: Empirical evidence of the benefits of workspace awareness in software configuration management. S. 113–123 of: SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. New York, NY, USA: ACM.

Sonnentag, Sabine – Niessen, Cornelia, – Volmer, Judith 2006: Expertise in Software Design, kirjassa The Cambridge Handbook on Expertise and Expert Performance. Cambridge Univ. Press. S. 373–387.

Tietojärjestelmien hankinta Suomessa 2013. 24.5.2013 ETLA. Tutkimusraportti. Saatavissa osoitteessa: <http://www.ttlry.fi/sites/ttl.ttlry.mearra.com/files/Tietoj%C3%A4rjestelmien%20hankinta%20Suomessa%202013.pdf>.

Vähäniitty, J. – Rautiainen, K. – Heikkilä, V. – Vlaanderen, K. 2010: Towards Agile Product and Portfolio Management. SPRG, Software Businesss and Engineering Laboratory, Aalto University.

Sivun 24 kuva: Juho-Pekka Virtanen.

**Kirjoittaja**

**Nikunen, Erja**, yliopettaja, FL. Metropolia Ammattikorkeakoulu, tietotekniikan koulutusohjelma, ohjelmistotekniikka.