Sytyke ry:n jäsenlehti
Nro ▪ 2/2011

SYSTEEMITYÖ

OHJELMISTOKEHITYKSEN ASIANTUNTIJA

s.10

Software Craftsmanship nostaa laadun ja osaamisen pöydälle

Systeemityön käsityöläiset

Sytykkeen laivaseminaari 2011 - ilmoittaudu jo! (www.sytyke.org)
Tsekkaa samasta osoitteesta myös uudet webbisivumme!

TIETOHALLINNON PROSESSIT SUJUVIKSI

HYÖDYNNÄ HAAGA-HELIAN LAAJA-ALAISTA LIIKETOIMINTA- JA IT-OSAAMISTA JA OHJAUSTA

Mahdollistamme organisaation tarpeista lähtevän toiminnan ja prosessien kuvaamisen, analysoinnin ja kehittämisen.

Palvelutarjontamme

- Selvitystyöt ja kartoitukset
- Mallinnukset ja analysoinnit
- Kehittämistyöpajat
- Kehittämisprojektit
- Koulutukset

http://prolab.haaga-helia.fi

Lisätiedot:
Jarmo Sarkkinen
jarmo.sarkkinen@haaga-helia.fi
050 412 6913

HAAGA-HELIA
ammattikorkeakoulu

Sytyttääkö? - Liity jäseneksi

Huomaa: *Liittyminen Sytykkeen jäseneksi maksaa vain 13€/vuosi, jos olet jo jonkin toisen TTL ry:n jäsenyhdistyksen jäsen! Jäsenenä saat muiden etujen lisäksi Systeemityö-lehden kotiin kannettuna neljästi vuodessa!*

Sytykkeen jäseneksi liitytään Tietotekniikan liiton kautta valitsemalla jäsenyhdistykseksi Systeemityöyhdistys ry. Nykyinen Tietotekniikan liiton jäsen voi liittyä joko vaihtamalla jäsenyhdistystä tai liittymällä lisäjäseneksi.

Tietotekniikan liiton henkilöjäsenmaksu vuonna 2011 on alkaen 53€ ilman lehtiä, yksi lehti 68€. Erityisryhmien hinnoittelusta lisätietoja Tietotekniikan liitosta. Lisäjäsenyys maksaa 13€/yhdistys.

Sytyke ry on vuodesta 1979 toiminut valtakunnallinen systeemityöntekijöiden ammatillinen yhdistys, joka kehittää alan ammattilaisten välistä yhteistyötä ja tutkimustoimintaa.

Teemayhdistyksen jäseneksi voivat liittyä kaikki systeemityöstä kiinnostuneet yksityiset henkilöt, yhdistykset ja yritykset. Sytyke ry:n toiminta-alueena on koko Suomi. Sytyke on Tietotekniikan liitto ry:n jäsenyhdistys.

Lisätietoja Sytyke ry:stä: www.sytyke.org

TOIMISTO
Sytyke ry
Tietotekniikan liitto ry
Lars Sonckin kaari 12
02600 Espoo
vaihde: 020 741 9898
fax 020 741 9889

www.ttlry.fi
jasenasiat@ttlry.fi

Julkaisija
Systeemityöyhdistys Sytyke ry
Tietotekniikan liitto ry
Lars Sonckin kaari 12
02600 Espoo
vaihde: 020 741 9898
fax 020 741 9889

Päätoimittaja
Juha Jääskinen
juha.jaaskinen@digia.com
puh. 050 420 0822

Toimitussihteeri
Anne Hakala

Taitto
Speaking Bark Tmi
Katja Tamminen
taitto.kt@speakingbark.com

Toimituskunta 2/2011
PrOSY, Mikko Holmberg

Lisätietoja lehdestä
www.sytyke.org/st

Tilaukset
Systeemityö-lehti sisältyy
Sytykkeen jäsenmaksuun.
Vuositilaus 30 €
Irtonumerot 8 €
min. 50kpl lisätilaus 2€/lehti
(viim. 1kk ennen julkaisupäivää).
Tilaukset yhdistyksen toimistosta.

Kansikuva
Katja Tamminen

2011 Numerot

Nro	Ilmestyy
1. Kokonaisarkkitehtuuri	09.03.2011
2. Systeemityön käsityöläiset	04.05.2011
3. Muurit murtuu	05.10.2011
4. Pilviteknologiat - year after	14.12.2011

Painopaikka
T-Print
Ahokaari 1-3
05460 Hyvinkää
Puh. (019) 475 8500

Painos: 3500 kpl
ISSN 1237-0525
18. vuosikerta, nro. 2

Ilmoitushinnat
Takakansi A4 1200 €
Sisäkannet A4 1000 €
Sisäsivut 1/1 800 €
Sisäsivut 1/2 600 €
Sisäsivut 1/4 400 €

Arvonlisävero 0%
Vakiopaikan vähintään vuodeksi
varanneille 20% alennus.

Kirjoituksia ja kuvia saa lainata
lehdestä vain toimituksen luvalla.
Toimitus on sitoutumaton eikä
ota vastuuta kirjoittajien
mielipiteistä. Toimitus ei
vastaa asiavirheistä.

Pääkirjoitus

Mikko Holmberg
Kirjoittaja on Sytyke ry:n varapuheenjohtaja ja toimii CodeBakers Oy:ssä TechnologyLead:na. Esitetyt mielipiteet ovat tekijän omia, eivätkä välttämättä edusta kirjoittajan työnantajanmielipiteitä.
[mikko.holmberg@codebakers.fi]

Tervehdys hyvät systeemityöläiset!

Tämänkertainen lehti syntyi halusta parantaa maailmaa - tai ainakin systeemityön maailmaa. Lehdessä on historiansa varrella käsitelty paljon erilaisia menetelmiä, arkkitehtuureja ja erilaisia tekniikoita, mutta tällä kertaa pyrimme tuomaan keskusteluun sen miten itse sovelluskehitystä voitaisiin parantaa.

Lehden otsikko "Systeemityön käsityöläiset" tulee Software Craftsmanship-liikkeestä, joka on tämän lehden pääteema. Liikkeen päämääränä on yksinkertaisesti tuottaa parempaa jälkeä ja näin enemmän lisäarvoa. Lue lehdestä lisää ajatuksista, miten tähän päästään.

Henkilökohtaisena lupauksena sanon sen, että ensi vuonna omalla puheenjohtajakaudellani Sytyke tulee ajamaan selkeämmin juuri toteuttajien asiaa ja kutsunkin muut mukaan talkoisiin!

Hyvää kevättä ja antoisia lukuhetkiä!

PS. kannesta löytyy yksi virhe. Löydätkö virheen?

---

3 ▪ **Pääkirjoitus**
Mikko Holmberg

4 ▪ **Olipa kerran kaksi IT-konsultointia tarjoavaa yritystä**
Henri Karhatsu ja Vesa Teikari

8 ▪ **Mikä maksaa?**
Mikko Holmberg

10 ▪ **Software Craftsmanship nostaa laadun ja osaamisen pöydälle**
Esko Luontola

14 ▪ **Scrum käyttöön – Nollasta sataan 16 viikossa?**
Raine Kauppinen ja Altti Lagstedt

19 ▪ **Seminaariuutisia**

20 ▪ **Mestari-kisälli -ajattelu nykyaikaisessa ohjelmistotyössä**
Peter Hägg

24 ▪ **Ohjelmistojen tuottajan ammattitaito, ylpeys ja laatu**
Erkki Pöyhönen ja Petteri Puurunen

26 ▪ **Kolumni: Kuutamolla**

Henri Karhatsu
Henri (KTM, FM/tietojenkäsittelytiede) on itsenäinen ohjelmistoalan yrittäjä osaamisalueinaan ketterät menetelmät, Java sekä Ruby on Rails. Parhaillaan hän toimii Nokialla Scrum Masterina.
[henri@karhatsu.com]

Vesa Teikari
Vesa (FM/tietojenkäsittelytiede) toimii ketterän sovelluskehityksen konsulttina Houston Inc:ssä.
[vesa.teikari@houston-inc.com]

Olipa kerran kaksi IT-konsultointia tarjoavaa yritystä

Softaprojekteja voidaan tehdä monella tavalla, mutta karkeasti ne voidaan jakaa kahteen tapaan: lyhytnäköisesti tai pitkäjänteisesti tuottoja tuottaviin. Mietitäänpä, miten tämä voisi mennä reaalimaailmassa.

Olipa kerran IT-konsultointia tarjoava yritys

Olipa kerran IT-konsultointia tarjoava Yritys, jonka tehtävänä oli maksimoida osakkeenomistajien varallisuutta. Kuten yrityksillä on tapana. Yrityksen liikevaihdosta pääosa tuli laskutuksesta, jonka Yrityksen konsultit saivat aikaan toimiessaan muiden yritysten alihankkijoina.

Yritys oli saanut jälleen kerran tilauksen eräältä suurelta asiakkaalta. Oli alkamassa projekti, johon Yritys oli onnistunut myymään resursseja. Projekti oli itse asiassa jo alkanut muutama kuukausi aiemmin määrittelyvaiheella, nyt vain tarvittiin muutama ohjelmoija toteuttamaan tehdyt suunnitelmat. Yrityksen myyntipäällikkö oli luonnollisesti mielissään, koska hän oli löytänyt projektin parille uudelle työntekijälle, joista toinen oli juuri päättänyt opiskelunsa ja oli näin valmis oikeisiin töihin. Tämä projekti sopi nuorelle työntekijälle oikein hyvin, koska kyse oli Java-projektista, ja Javasta hänellä oli kokemusta jo opiskeluajoilta.

Ohjelmoijien lisäksi Yritys oli myynyt projektiin projektipäällikön. Yhdessä asiakkaan kanssa projektipäällikkö oli päätynyt siihen, että ohjelma voidaan julkaistaan neljän kuukauden päästä. Yksi vanhemmista ohjelmoijista ei pitänyt tätä kovin realistisena mutta perustuen aiempiin kokemuksiinsa ei siitä myöskään erityisesti kenellekään sanonut. Toinen ohjelmoija otti asian puheeksi projektipäällikön kanssa, joka kuitenkin torjui epäilyt, koska kysehän oli asiakkaan päätöksestä.

Projekti sujui mukavasti monta viikkoa. Tai tarkkaan ottaen ohjelmoijat eivät aivan ymmärtäneet, mitä vaatimukset tarkoittivat, mutta yhdessä projektipäällikön kanssa he pystyivät kuitenkin tulkitsemaan määrittelyitä. Kun asiakas kysyi ensimmäisen kuukauden jälkeen projektipäälliköltä projektin etenemisestä, tämä vastasi tiimin etenevän suunnitelmien mukaan. Hieman häntä kuitenkin epäilytti tavoitteen saavuttaminen. Myyntipäällikkö oli konsultteihin yhteydessä kuukauden viimeisenä päivänä, jotta he muistasivat lähettää tiedot tehdyistä tunneista Yrityksen assistentille.

Projekti kohtasi ensimmäisen merkittävän haasteen, kun asiakas oli saanut palautetta omilta asiakkailtaan. Palaute vaikutti kehitettävään ohjelmaan siten, että alkuperäisiin suunnitelmiin piti tehdä muutoksia. Tämä luonnollisesti huolestutti projektipäällikköä, koska se heikensi tiimin mahdollisuuksia saavuttaa julkaisutavoite. Erityisesti hän säikähti ohjelmoijien arviota siitä, kuinka paljon muutoksen tekeminen kestäisi. Onneksi yksi ohjelmoijista kuitenkin totesi, että muutos voidaan tehdä myös kopioimalla osa koodista uuteen moduuliin. Tämä säästäisi aikaa ja lisäksi varmistaisi paremmin sen, ettei jo valmiiksi tehtyä koodia rikottaisi.

Mitä pidemmälle projekti eteni, sitä selkeämmäksi tuli, ettei neljän kuukauden tavoitteeseen päästä. Tässä vaiheessa yksi tiimin jäsenistä alkoi kyllästymään asiakkaan tiettyihin toimintatapohin, mutta projektipäällikkö huomautti, kuinka asiakas loppukädessä asioista päättää. Projektipäällikkö myös muistutti naurahtaen, että samanlaistahan tämä konsultin työ on aina ollut, eikä se siitä mihinkään tule muuttumaan.

Yhtä tiimin jäsentä ärsytti nuorimman ohjelmoijan osaamattomuus. Vanhempi ohjelmoija oli vastikään joutunut muokkaamaan nuoremman koodia, koska heidän vastuullaan olleiden komponenttien täytyikin ennakkosuunnitelmista poiketen kommunikoida keskenään. Oli paljastunut, että nuorempi ohjelmoija oli jo pitkään käyttänyt tietorakenteita kyseenalaisella

tavalla. Lisäksi hän ei ollut osannut höydyntää rajapintoja oikein. Vanhemman ohjelmoijan kannalta hyvä asia oli se, ettei hänen tarvinnut tämän muutoksen jälkeen katsoa nuoremman koodia, sillä jokainen pystyi kuitenkin työskentelemään pääsääntöisesti oman komponenttinsa parissa. Tämän hän myös otti esille, kun nuorempi ehdotti varovasti, että tiimin ehkä kannattaisi sopia yhteisistä säännöistä koodin suhteen. Turhaahan se oli, kun kerran jokainen vastasi omasta koodistaan.

Ensimmäisen julkaisun lähestyessä paine tiimin ympärillä kasvoi. Projektipäällikön lisäksi asiakas toi esille useita muutos- ja korjaustarpeita. Tiimi teki muutoksia niin nopeasti, kun pystyi, mutta aika usein jo kerran korjatut ongelmat tulivat esille uudestaan. Tämä turhautti asiakasta, joka joutui testaamaan samoja toiminnallisuuksia moneen kertaan. Ohjelmoijilla olisi ollut vuotuinen koulutuspäivä, mutta projektipäällikkö ja myyntipäällikkö yhdessä päättivät, että sitä on syytä lykätä syksyyn, koska projektissa oli parhaillaan niin kova kiire. Yksi ohjelmoijista oli oikeastaan aikonut ehdottaa, että koulutusta pitäisi olla useamminkin, mutta jätti ottamatta asian esille kuullessaan tehdyn päätöksen. Sitä paitsi kun hän oli asiaa hieman tiedustellut edellisissä pikkujouluissa, ei Yrityksen toimitusjohtaja ollut kovin innostunut, sillä monen päivän koulutukset olisivat vähentäneet merkittävästi konsulttien laskutusta.

Lopulta ohjelma saatiin julkaistua, vaikkakin kaksi kuukautta myöhässä. Itse asiassa voitiin sanoa, että projekti oli mennyt normaalia paremmin, koska usein myöhästymiset olivat olleet paljon suurempia. Tässä vaiheessa oli joka tapauksessa selvää, että ohjelmaan tarvittiin myös paljon lisäominaisuuksia, koska alkuperäiset määrittelyt olivat menneet jonkin verran pieleen. Lisäksi oli tarvetta jatkuvalle ylläpitotyölle, koska tuotteesta löytyi käytön aikana paljon korjattavaa. Tämä oli tietenkin normaalia, koska kysehän on alasta, jossa harvoin onnistutaan tekemään kerralla toimivaa ohjelmistoa.

Ensimmäisen neljän kuukauden jälkeen konsulttien sopimuksia oli jatkettu kolmella kuukaudella, jotta ohjelma saadaan valmiiksi. Myyntipäälliköllä oli kuitenkin helppo työ neuvotella tämän päälle vielä ainakin kuusi kuukautta lisää, sillä tekemistähän riitti. Hieman harmaita hiuksia aiheutti yhden ohjelmoijan lähtö toiseen yritykseen hänen kyllästyttyä lopullisesti työympäristöönsä. Toisaalta ei asia ollut mikään ongelma myyntipäällikön näkökulmasta, sillä hänellä oli tarjota asiakkaalle korvaava resurssi. Tilanne sitä paitsi muuttui Yrityksen näkökulmasta paremmaksi, sillä uudella ohjelmoijalla oli pienempi palkka kuin pois lähteneellä, mikä paransi Yrityksen katetta. Kun uusi työntekijä aloitti projektissa, muistutti myyntipäällikkö kuukauden viimeisenä päivänä, että työntekijän pitää lähettää tehdyt tunnit Yrityksen assistentille, jotta Yritys voi laskuttaa asiakasta.

Olipa kerran IT-konsultointia tarjoava yritys

Olipa kerran IT-konsultointia tarjoava Yritys, jonka tehtävänä oli maksimoida osakkeenomistajien varallisuutta. Kuten yrityksillä on tapana. Yrityksen liikevaihdosta pääosa tuli laskutuksesta, jonka Yrityksen konsultit saivat aikaan toimiessaan muiden yritysten alihankkijoina.

Yritys oli saanut tilauksen eräältä suurelta asiakkaalta. Asiakkaan oli ollut tarkoitus ensin määritellä tuote ja sen jälkeen aloittaa toteuttaminen, mutta Yrityksen myyntipäällikkö ei pitänyt tätä järkevänä lähestymistapana. Hän sen sijaan ehdotti, että asiakas ottaisi ohjelmoijat mukaan heti alkuvaiheessa, jotta kaikki osapuolet saavat parhaan käsityksen siitä, mitä ollaan tekemässä. Tähän asiakas varovasti suostui, kun oli ensin kuunnellut myyntipäällikön perustelut.

Myyntipäällikkö oli mielissään, koska oli löytänyt uudelle, nuorelle työntekijälle projektin. Työntekijä oli rekrytoitu yritykseen nuoresta iästään huolimatta, koska hänen avoimen lähdekoodin sovelluksiin tekemä kontribuutio oli vakuuttanut Yrityksen teknologiajohtajan. Ennen projektin alkua sovittiin, että yksi tiimin jäsenistä ottaisi nuoren työntekijän vastuulleen ja toimisi tämän mentorina.

Asiakas oli kysynyt Yritykseltä projektiin myös projektipäällikköä, mutta sen sijaan Yritys tarjosi Scrum Masteria. Tämänkin osalta myyntipäällikkö kävi keskustelun asiakkaan kanssa siitä, miten tällainen rooli eroaa

projektipäällikön roolista ja mitä Scrum Masterin odotetaan tekevän. Asia alkoi varovasti konkretisoitua asiakkaalle, kun Scrum Master ehdotti toista lähestymistapaa asiakkaan ehdotukseen neljän kuukauden julkaisutavoitteesta. Sen sijaan, että määritettäisiin yksityiskohtaisesti ominaisuudet, jotka tuossa vaiheessa pitäisi olla, luotiin visio tuotteen toiminnasta neljän kuukauden päähän ja julkaistaisiin ensimmäinen versio jo kahden viikon kuluttua. Tiimin ollessa vahvasti mukana esittämässä näkemyksiään siitä, minkälaisia ominaisuuksia tuotteeseen voitaisiin saada neljässä kuukaudessa, asiakas suostui ehdotukseen.

Projektin ensimmäiset viikot sujuivat mukavasti. Ohjelmoijat kävivät jatkuvaa keskustelua asiakkaan kanssa tuotteen ominaisuuksista ja toivat myös oman näkemyksensä vahvasti esille. Aluksi tämä hämmensi asiakasta, koska perinteisesti oli toimittu hieman eri tavalla ja varsinkaan alihankkijoilla ei ollut ollut tapana ottaa kantaa näin selvästi. Asiakas kuitenkin alkoi luottaa alihankkijoihin vähitellen, koska hän näki hyvin nopeasti uusia ja toimivia versioita tuotteesta.

Projekti kohtasi ensimmäisen merkittävän haasteen, kun asiakas oli saanut palautetta omilta asiakkailtaan, jotka olivat käyttäneet ensimmäisiä versioita ohjelmasta. Palaute vaikutti kehitettävään ohjelmaan siten, että alkuperäisiin suunnitelmiin piti tehdä isoja muutoksia. Asiakas oli huolissaan siitä, että tämä pysäyttäisi kehitystyön tai ainakin hidastaisi sitä merkittävästi. Hän vaati, että tiimi tekisin nopeimman mahdollisimman ratkaisun, jolla ongelmasta päästään eroon. Ohjelmoijat eivät tähän kuitenkaan halunneet suostua, sillä heille oli päivänselvää, kuinka kriittistä tuotteen sisäinen laatu on pitkällä aikavälillä. He halusivat kaikin keinoin välttää koodin huonoa luettavuutta ja monimutkaisuutta, toisteista koodia sekä ylimääräisiä riippuvuuksia. Erityisen tärkeää heille oli, että tuotantokoodia tukee kattava määrä testikoodia, joka varmistaa sen, etteivät muutokset riko muita osia ohjelmasta.

Scrum Master tunnisti potentiaalisen konfliktin ja järjesti yhteisen keskustelutilaisuuden asiakkaan ja ohjelmoijien kesken. Asia käytiin avoimesti läpi niin, että molemmat osapuolet pystyivät tuomaan esille oman näkemyksensä ja ymmärsivät näin paremmin toisiaan. Tämä lisäsi luottamusta osapuolten välillä. Keskustelun seurauksena tiimi löysi helpomman tavan toteuttaa sama loppukäyttäjän tarve, ja asiakas oppi ymmärtämään paremmin, mitä laadukas koodi tarkoittaa. Tiimi käytti muutoksen tekemiseen kaksi viikkoa ja palasi sen jälkeen normaaliin rytmiin uusien ominaisuuksien tekemisessä. Asiakas oli itse asiassa hieman yllättynyt siitä, ettei iso muutos ollut rikkonut muita osia tuotteesta, kuten aiemmissa projekteissa oli usein käynyt.

Projektin alusta alkaen ohjelmoijat olivat jakaneet vastuuta niin, ettei kenellekään syntynyt erityisiä erikoisalueita. Jokaisen muutoksen yhteydessä tehtiin välitön koodikatselmointi, jolla varmistettiin yhteisesti sovittujen käytäntöjen noudattaminen ja joka antoi erityisesti nuorimmalle ohjelmoijalle arvokasta palautetta. Lisäksi hän kävi mentorinsa kanssa jatkuvaa keskustelua omasta ja tiimin toiminnasta ja pääsi näin vahvasti sisälle siihen, kuinka menestyvä ohjelmistokehitystiimi toimii ja erityisesti, mitä tarkoittaa laadukas koodi. Kävipä pari kertaa niinkin päin, että nuorempi ohjelmoija pystyi esittämään parhaan ratkaisun tiettyyn ongelmaan, mikä oli hyödyllistä myös mentorin oppimisen kannalta.

Tiimi pystyi etenemään tasaiseen tahtiin. Vaikka kaikilla oli alkuperäinen visio kirkkaana mielessä, oli asiakas itse asiassa jo unohtanut alkuperäisen neljän kuukauden tavoitteen. Oli nimittäin käynyt niin, että tuote oli jo julkaistu kaikkien loppuasiakkaiden käytettäväksi kahden kuukauden jälkeen projektin aloituksesta. Siinä ei ollut lähellekään kaikkia niitä ominaisuuksia, joita asiakas oli alunperin ajatellut saavansa neljässä kuukaudessa, mutta tuote oli kuitenkin ehdottomasti käyttökelpoinen. Sitä paitsi ei asiakas enää kaivannut alkuperäisiä suunnitelmia, sillä oppiessaan lisää tuotteen käyttäjien tarpeista tiimi oli pystynyt tarjoamaan järkevämpiä tapoja toteuttaa käyttäjien tarpeet.

Kehitystyötä ei hidastanut merkittävästi edes se, että kaikki ohjelmoijat olivat vähintään päivän kuukaudessa poissa kehitystyöstä. Tämän ajan he käyttivät omaan oppimiseensa niin, että useimmiten Yrityksen muut työntekijät esittelivät kiinnostavia uusia tekniikoita, tai sitten he yhdessä viettivät aikaa harjoitellen erilaisia ohjelmointikäytäntöjä. Eräässä seminaarissa asiakas kysyi Yrityksen toimitusjohtajalta siitä, kuinka paljon jatkuva kouluttaminen maksaa Yritykselle. Toimitusjohtaja vastasi, että oleellisempi kysymys on se, kuinka paljon maksaisi olla tarjoamatta jatkuvaa koulutusta.

Konsulttien alkuperäiset sopimukset olivat olleet neljän kuukauden mittaisia. Ennen kuin Yrityksen myyntipäällikkö ehti jutella asiakkaan kanssa mahdollisesta jatkosta, otti asiakas yhteyttä häneen. Tuotteen käyttäjämäärä kasvoi sen verran vahvasti, että asiakas oli päättänyt jatkaa tuotteen kehittämistä edelleen ja näin ollen halusi myös konsulttien sopimuksia jatkettavan. Lisäksi asiakas kertoi myyntipäällikölle uudesta projektista, johon hän halusi tiimin nimenomaan Yritykseltä. Sen verran vakuuttunut hän oli Yrityksen työntekijöiden aikaansaannoksista ensimmäisen projektin parissa. ■

Osaamisyhteisöt 2011

Sytykkeessä toimitaan niin yhdistystasolla kuin aihepiireittäin erikoistuneissa osaamisyhteisöissä. Monipuolisessa tarjonnassamme löytyy jokaiselle jotakin. Vaihtoehtona on myös perustaa omalle kiinnostukselleen uusi osaamisyhteisö – Sytyke-hallitus toivottaa toimintaehdotukset tervetulleeksi. Osaamisyhteisön toimintaan pääset mukaan laittamalla postia vetäjälle.

BisOSY
BisOSY on ohjelmistoliiketoiminnan osaamisyhteisö, jonka tavoitteena on parantaa suomalaista ohjelmistoliiketoimintaosaamista ja kiihdyttää alan kasvua ja kansainvälistymistä.
Vetäjänä Jyrki Kontio. [jyrki.kontio@iki.fi]

Ketterän kehityksen OSY
Ketterän kehityksen osaamisyhteisössä pääset verkottumaan ketterästä ohjelmistokehityksestä kiinnostuvien muiden ammattilaisten kesken. Osaamisyhteisölle ollaan keräämässä aktiivista ohjausryhmää.
Vetäjänä toimii Erkki Laitila. [erkki.laitila@swmaster.fi]

KäytettävyysOSY
KäytettävyysOSY tarjoaa jäsenilleen mahdollisuuden verkostoitua ja vaihtaa kokemuksia käytännössä toimiviksi osoittautuneista käytettävyyden kehittämiskeinoista sekä kehittää omaa käytettävyysosaamistaan. Vetäjänä ja yhteyshenkilönä Matti Vuori.
[xmvuori@kolumbus.fi]

MallinnusOSY
MallinnusOSYn tehtävä on yhdistää ohjelmistokehitykseen liittyvästä mallinnuksesta kiinnostuneet henkilöt keskustelemaan ja vaihtamaan ajatuksia sekä välittää tietoa aiheesta. DAMA (www.damafinland.org) on yhdistetty MallinnusOSYyn.
Vetäjänä toimii erikoistutkija Juha Pärssinen VTT:ltä.
[juha.parssinen@vtt.fi] (puh. 020 722 5612)

PrOSY
Projektitoiminnan osaamisyhteisö yhdistää systeemityöprojektien kehittämisestä kiinnostuneet. Vetäjänä Petteri Puurunen.
[petteri.puurunen@tieto.com]

RELA
on vapaamuotoinen, relaatiotietokantoihin keskittynyt osaamisyhteisö, joka kokoontuu 1-2 kuukauden välein. Tapaamisissa on aina pari alustajaa tai esittäjää, jotka ovat yleensä kerhon jäseniä.
Vetäjänä Lauri Pietarinen.
[lauri.pietarinen@relational-consulting.com]

SK SIG - Suomen suorituskykytestaajat
SK SIG Suomen suorituskykytestaajat on tietojärjestelmien suorituskykyyn keskittyvä TestausOsyn ja RELA:n rinnakkaisryhmä.
Vetäjänä toimii Ilkka Myllylä. [ilkka.myllyla@reaaliprosessi.com]

SOA SIG
SOA SIG (Special Interest Group) on palveluarkkitehtuuriin (Service-Oriented Architecture, SOA) keskittynyt osaamisyhteisö, jonka tarkoituksena on tarjota jäsenilleen mahdollisuuksia vaihtaa kokemuksia ja tietoa palvelulähtöisten liiketoiminta-arkkitehtuurien periaatteista, menetelmistä, välineistä ja sovelluksista.
Vetäjänä Janne J. Korhonen. [janne.korhonen@jannekorhonen.fi]

TestausOSY - FAST
TestausOSY on testauksen parissa työskentelevien yhteinen verkosto. Testauksen osaamisyhteisön tarkoituksena on edistää suomalaisen ohjelmistotestauksen ja ohjelmistotestaajien asemaa ja auttaa kaikkia testauksen parissa painiskelevia. Paikallistoimintaa on pääkaupunkiseudulla, Tampereella sekä Oulussa. Vetäjänä toimii Maaret Pyhäjärvi. [maaret.pyhajarvi@iki.fi]

KAOS
Kokonaisarkkitehtuurin osaamisyhteisö, joka pyrkii edistämään liiketoimintalähtöistä kokonaisarkkitehtuurinäkemystä suomalaisissa yrityksissä ja yhteisöissä, muodostamaan aktiivisen arkkitehtiverkoston sekä synnyttämään kokonaisarkkitehdin ammatti-identiteetin.
Vetäjänä toimii Anna Aaltonen. [anna.aaltonen@fortum.com]

Hyvä tietotekniikan liiton jäsen!

TTL Tietotekniikan liitto ry

Näin päivität tietosi

Voit päivittää jäsentietosi verkkosivuillamme www.ttlry.fi. Tietojen päivittämiseen tarvitset käyttäjätunnuksen (= jäsennumerosi, merkitty jäsenlehtiin) ja salasanasi (= postinumerosi). Jos olet muuttanut salasanasi tai kirjautuminen ei muutoin onnistu, voit lähettää tunnusten tarkistuspyynnön osoitteella jasenasiat@ttlry.fi.

Toivomme sinun erityisesti varmistavan, että sähköpostiosoitteesi jäsentiedoissa on oikea. !

Henkilökohtaisempaa palvelua - Sinun eduksesi

Tietotekniikan liitto jäsenyhdistyksineen, osaamisyhteisöineen ja kerhoineen haluaa palvella jäseniään henkilökohtaisemmin ja paremmin, tarjota tietoa juuri Sinua kiinnostavista aiheista. Palvelun parantamiseksi olemme uusineet verkkopalvelumme.

Päivität vain tiedot itseäsi kiinnostavista aiheista ja saat tietoa juuri niistä. Voit päivittää valintasi aina halutessasi. Tietoja ei anneta ulkopuolisille tahoille vaan niitä käytetään ainoastaan TTL:n ja sen piirissä toimivien yhteisöjen tarkoituksiin.

Tietotekniikan liitto ry
Lars Sonckin kaari 12
02600 Espoo
www.ttlry.fi
etunimi.sukunimi@ttlry.fi
jasenasiat@ttlry.fi
p. 020 741 9898
f. 020 741 9889

Mikko Holmberg
Kirjoittaja on Sytyke ry:n varapuheenjohtaja ja toimii CodeBakers Oy:ssä TechnologyLead:na. Esitetyt mielipiteet ovat tekijän omia, eivätkä välttämättä edusta kirjoittajan työnantajanmielipiteitä.
[mikko.holmberg@codebakers.fi]

Ohelmistojen valmistus vaatii rahaa ja aikaa. Rahaa yleensä aikalailla. Kustannuksia on totuttu optimoimaan toteutusvaiheessa, mutta jos tarkastellaan sovelluksen koko elinkaarta, on sen kallein osa ylläpito. Mikä tässä oikeen maksaa? Miten voitaisiin säästää? Software Craftsmanship – systeemityön käsityöläisyys pyrkii omalta osaltaan ratkomaan ongelmaa.

Ohjelmistokehitys on aikavievää ja kallista. Monet menetelmät ja prosessit paneutuvat alhaisiin kustannuksiin ja toteutusajan optimointiin. Esimerkiksi ketteryyden yksi pääperiaate on ollut kustannussäästöt reagoimalla muutostarpeisiin mahdollisimman pian. Eli mitä myöhemmin kehityssyklissä huomataan muutostarve tai virhe, sen kalliimpaa on sen toteuttaminen. Perinteisemmissä menetelmissä puolestaan pyrittiin arvioimaan kustannukset ja aikataulu etukäteisolettamien perusteella. Oltiin mitä mieltä tahansa näiden lähestymistapojen paremmuudesta, jää kuitenkin huomioimatta yksi tärkeä seikka. Sovelluksen kustannukset sen koko elinkaaressa muodustuvat 60-80%:ti ylläpidosta*. Järjestelmien tilaajat ja varsinkin isommat toimittajat tämän toki tietävät, mutta se ei silti näy itse toteutusvaiheessa.

Kun sovellusta tai laajempaa tietojärjestelmää rakennetaan optimoidaan aikataulua ja kustannuksia. Kaikki osapuolet ovat tyypillisesti tyytyväisiä, jos tietojärjestelmäprojekti on pysynyt suunnitellussa aikataulussaan ja sen sisältö on sitä, mitä tilaaja lopulta halusi. Onko järjestelmä silti kustannustehokas? Jos verrataan ylläpidettävää ja selkää järjestelmään nopeasti toteutettuun "spagettikoodi"-järjestelmään, vastaus lienee selkeä. Donal C. Martin esittää asian liittyen spagettikoodin suhteesta tuottavuuten, joka selkiää kuvassa 1 esitetyssä kuvaajassa - "The Total Costs of Owning a Mess: Productivity vs. time"[1].

Miksi ylläpitovaihe on sitten yleensä kallis? Toki luonnollinen selitys on se, että vaihe on pituudeltaan moninkertainen verrattuna toteutusvaiheessa. Toisaalta ylläpidossa käytetään yleensä huomattavasti vähemmän työvoimaa verrattuna itse toteutukseen. Kokonaan ei siis ylläpitovaiheen pituus selitä kustannuksia. Väitän, että kustannukset syntyvät huonosta koodista ja sen rakenteesta.

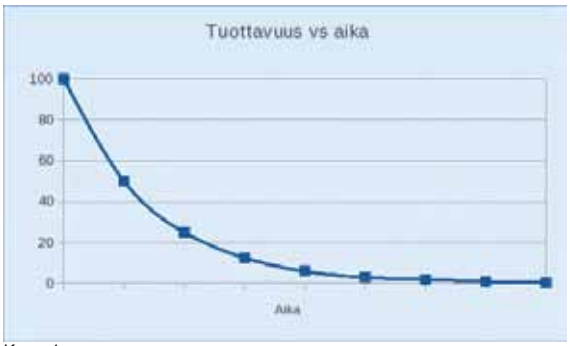
Miksi huonoa koodia ylipäätänsä tehdään? Helppo vastaus tähän on tietysti se, että osaaminen on riittämätöntä. Toinen tärkeä syy on toteutusvaiheen optimointi. On hyvin vaikea myydä ajatusta, että toimiva koodi ei ole vielä valmista. Ajattele itseäsi projektipäällikkönä, joka kuulee että ominaisuus on toteutettu ja se on läpäissyt kaikki hyväksymistestit, mutta se ei ole vielä valmis. Antaisiko räätäli tekemänsä takin tilaajalle, jos se näyttää hyvältä, mutta sisäpuoli on viimeistelemätön?

Miksei puolestaan osaamiseen olla puututtu? Monella yrityksellä on käytettävissään todella kokeneita osaajia, jotka eivät osaa tuotaa ylläpidettävää ja ymmärrettävää koodia. Tämä johtuu taas siitä, että tätä ei vaadita, yrityksessä ei välttämättä ole kulttuuria tähän tai sitten vaan ei välitetä. Lisäksy myös

tuttu juttu on se, että kokeneimmista osaajista tulee jotain muuta kuin kokeneita koodareita. Osaaminen liittyy osittain uraan ja osittain henkilöstön hallintaan. Toisaalta ongelma on itse ohjelmoijien asenteessa ja toisaalta heitä johtavissa henkilöissä. Monen kunnianhimoisen uraihmisen on vaikea nähdä sitä, että useampi koodari haluaa vain tulla paremmaksi koodariksi – ei välttämättä projektipäälliköksi tai muuksi johtajaksi. Tähtääkö esimerkiksi kirurgiaan erikoistuva lääkäri lopulta hallinnolliseksi lääkäriksi? Joku ehkä, mutta suurin osa ei. Haasteena tässä puolestaan johdolle on tarjota tarpeeksi palkitseva eteneminen asiantuntijauralla. Lisäksi esimiesten tulee olla tietoisia itse työstä ja sen haasteista. Monella esimiehellä on kovin virheellinen käsitys siitä, että jonkin kielen tai tekniikan osaaminen yksin tekisi osaajasta hyvän. Sepän on hyvä tuntea, miten rauta käyttäytyy, mutta se ei tee hänestä vielä hyvää seppää.

En tunne yhtään ohjelmoijaa, joka tietoisesti haluaisi tuottaa huonoa koodia. En tunne myöskään montaa (jos yhtään), joka ei haluaisi kehittyä ammatissaan. Ongelma vain on monesti siinä, että itse työ vie kaiken ajan ja kaikki pitää hoitaa mahdollisimman nopeasti. Nopeimmat palkitaan ja laatu mitataan vain vaatimuksien toteuttamisella.

Miten kustannuksia saataisiin alennettua? Ensisijainen asia tämän ratkaisemiseksi on ymmärrys siitä, että mahdollisimman nopeasti tehty työ ei ole mahdollisimman edullinen. Ei tämä tietysti toimi myöskään käänteisesti, eli mahdollisimman hitaasti tehty työ olisi mahdollisimman edullinen. Tässä yhteydessä pitää varmasti luodan jonkinlaisen koodin laadun kriteerejä ja niitä on jo aika lailla tarjolla. Tällaisia ovat esimerkiksi yksikkötestien kattavuus, staattisten koodianalyysien tulokset ja erilaiset muut (esim. arkkitehtuuri-) analyysit.



Kuva 1.

Toinen ongelma, johon pitää paneutua on osaaminen ja sen kehittäminen. Helpointa tämän edistäminen on itse työn parissa. Monet extreme programming (XP)-käytännöt auttavat tässä. Lisäksi mahdollisimman heterogeenisten tiimein käyttö on loistava keino edistää osaamisen leviämistä. Ei pelkästään niin, että aloittelijat oppivat konkareilta vaan oppimista tapahtuu myös päinvastoin. Aloittelijoilla ei ole yleensä fakkiutuneita ajatusmalleja, joita monelta konkarilta löytyy ja tällöin voidaan löytää aivan uusia ongelmanratkaisutapoja. Lisäksi on selvää, että osaamisen kehittämiseen ja ylläpitoon vaaditaan paljon muutakin kuin itse työssä oppimista. Tähänkin on monet yritykset ja yhteisöt jo lähteneet esim. erilaisten työpajojen avulla. Erittän hyvä työpajamuoto ohjelmointitaitojen kehittämiseen on ns. Coding Dojo jossa tiimi ratkaiseen jonkin ongelman pari kerrallaan muiden seuratessa ja kommentoidessa etenemistä. Kolmas tapa, joka on osoittautunut hyväksi oppimiskeinoksi, on vertaiskatselmoinnit. Tällöin toteuttajat käyvät ryhmässä läpi koodin ja kommentoivat sen ratkaisuja.

Tässä lehdessä puhutaan liikkeestä, joka on lähtenyt muutamia vuosia sitten liikkeelle edelläkuvatuista ongelmista johtuen. Itse ideat ovat vanhoja, perustuvat vanhoihin oppeihin ja ajatuksiin, mutta Software Craftsmanship-liike puhaltanut uutta tuulta näihin ideoihin. Tässä lehdessä viitataan ko. liikkeen alkuvoimiin, joten en tässä yhteydessä viittaa niihin sen kummemmin. Toivon, että asiaan ei suhtauduttaisi – puoleen tai toiseen – kovalla kiihkolla, vaan se ymmärrettäisiin enemmän haluna tehdä asiat paremmin.

Kirjoitin tämän artikkelin lähinnä Software Craftsmanship:n näkökulmasta ja omista havainnoistani sovelluskehitystyöstä. Ennen artikkelin julkaisua sain palautetta, että jonkinlaisia viitteitä olisi hyvä esittää. Pääsääntöisesti esittämäni asiat ovat omia mielipiteitäni, mutta en voinut olla etsimättä viitteitä. Englanninkielisiä hakutuloksia oli hankala käsitellä niiden runsauden (ja mahdollisesti haun huonouden) johdosta, joten tein suomenkielisen haun. Törmäsin vuonna 2007 tehtyyn dosenttityöhön[2]. Aihe on edelleen ajankohtainen. ■

** Tästä on vaikea löytää yleistäviä tutkimuksiani. Oma tulkintani kärjistää hiukan erilaisia haarukoita, joiden alapää on n. 50:ssä prosentissa ja yläpää lähes 80:ssä prosentissa. Viesti kuitenkin on, että puhutaan valtaosasta kustannuksia.*

Viitteet

1 *Martin, Robert C, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008*

2 *Koskinen, Jussi, dosenttityö, Jyväskylän yliopisto 2008; http://www.slidefinder.net/o/ohjelmistojen_yll%C3%A4pito_jussi_koskinen_ktt/9982369*

Esko Luontola

Kirjoittaja työskentelee ohjelmoijana ja käyttöliittymäsuunnittelijana. TDD-kärpänen puri häntä vuonna 2007 ja hän pyrkii edistämään craftsmanship-ajattelua minne menekään. [www.orfjackal.net]

Software Craftsmanship nostaa laadun ja osaamisen pöydälle

Software Craftsmanship on ohjelmistojen laatuun ja ohjelmistokehittäjien osaamiseen huomiota kiinnittävä liike. Se on yritys herättää ihmisissä intohimoa, kurinalaisuutta ja vastuuntuntoa tekemisiään kohtaan. Kehittäjien pitäisi panostaa oman osaamisensa jatkuvaan parantamiseen ja parhaansa tekemiseen, niin että heidän tuotoksiaan olisi ilo käyttää ja jatkokehittää.

Tuho

Software Craftsmanship on noussut vastapainoksi sille, että Agile-keskusteluissa huomio on viimevuosina siirtynyt enenemässä määrin pois teknisistä toteutusasioista ja sen sijaan keskitytään vain prosesseihin ja projektinhallintaan. Scrum ja Kanban puhuttavat, Extreme Programmingia tuskin kuulee mainittavan.

Useat firmat väittävät olevansa "ketteriä", mutta todellisuudessa niissä tuotetaan samaa sontaa kuin ennenkin - nyt sitä vain tehdään "sprinteissä". Monien tiimien toteutusmenetelmät ja tekninen osaaminen eivät yllä sille tasolle, että he pystyisivät tuottamaan laadukasta jälkeä nopeasti ja kestävästi, vaan ennen pitkään koodin laatu alkaa rakoilla, uusien toimintojen lisäämiseen kuluu yhä enemmän aikaa ja aikaa tuhlaantuu bugien paikkailuun.

Kuten Dave Thomas, yksi Agile Manifeston allekirjoittajista, sanoi muistelmissaan 10 vuotta manifeston kirjoittamisen jälkeen[1], *ketteryys* on adjektiivi. Ketteryys tarkoittaa kykyä liikkua nopeasti ja helposti. Kuitenkin monet ajattelevat sitä substantiivina ja sanovat *käyttävänsä Agilea* (huomaa iso alkukirjain) noudattaessaan joitakin tiettyjä prosesseja ja menetelmiä. Jatkuvan kehittymisen ja uusien asioiden kokeilun sijaan he keskittyvät valmiiksi paketoidun eineksen jäljittelyyn. Dave Thomas sanookin, että ei ole mahdollista "tehdä Agilea" ja sitä kautta tulla paremmaksi, vaan on ainoastaan mahdollista tehdä asioita paremmin ja sitä kautta tulla ketteräksi.

Synty

Yksi Software Craftsmanshipin alkukipinöistä leimahti vuonna 2001, kun Pete McBreen esitti kirjassaan *Software Craftsmanship: The New Imperative* näkemyksensä ohjelmistokehityksestä: rakennusteollisuus ei metaforana vastaa ohjelmistokehitystä, vaan entisaikojen koko ikänsä taitojaan hioneet käsityöläiset ovat parempi metafora. Myös sitä aiemmin, vuonna 1999 julkaistu *The Pragmatic Programmer: From Journeyman to Master* esitti vastaavan ideologian jatkuvasta osaamisensa kehittämisestä ja laadukkaan työn tekemisestä. Jo kirjan kannessa olevan käsityöläisen puuhöylän voi yhdistää McBreen sittemmin kärjistämään craftsmanship-metaforaan. Ja vielä aiemmin, muun muassa vuoden 1986 *No Silver Bullet* -kirjoituksessa, on ollut tiedossa hyvien ja erinomaisten ohjelmistokehittäjien välillä oleva *kymmenkertainen* tuottavuusero, jota myös McBreen kirja käsittelee. Itse asiassa No Silver Bullet -kirjoituksen viimeiset kaksi sivua ovat kuin suoraan Software Craftsmanship -kirjasta – osaamista vastaavaa palkkaa myöten.

Viimevuosina Software Craftsmanshipin äänekkäimpiä puolestapuhujia on ollut Robert "Uncle Bob" Martin, jonka mieluisimpiin puheenaiheisiinsa konferensseissa kuuluu ohjelmoijien kannustaminen oppimaan parempia toteutustason ohjelmistoke-

hitysmenetelmiä, kuten Test Driven Development, ja noudattamaan partiopoikasääntöä - *aina kommitoi lähdekoodi versionhallintaan siistimpänä kuin sen saadessasi* - mistä kaikesta hän käytti termiä *professionalism*.

Mielestäni tätä *professionalism/craftsmanship* -ajattelutapaa hyvin kuvaava lausahdus on vuoden 2007 JAOO-konferenssissa pidetyssä haastattelussa[2], missä heräsi kysymys että mitä tehdä jos pomo ei anna käyttää riittävästi aikaa hyvän koodin kirjoittamiseksi. Pete McBreen letkautti vastaukseksi, että silloin on aika huijata. Robert Martin puolestaan sanoi, ettei pidä sitä edes huijaamisena, vaan ammattilaisena olemisena; projektipäällikölle ei pitäisi tarvita sanoa, että "teen erityisponnisteluja kirjoittaakseni siistiä koodia ja sen vuoksi ylitän aikarajani", vaan sen kuin vain tekee koodistaan siistiä. Kuten hän tapaa sanoa, ainoa keino edetä nopeasti on tehdä hyvää jälkeä.

Ainoa keino edetä nopeasti on tehdä hyvää jälkeä.

- Robert C. Martin

[http://programmer.97things.oreilly.com/wiki/index.php/Speed_Kills]

Agile 2008 -konferenssin avainpuheessa Robert Martin ehdotti viidennen arvon lisäämistä Agile-manifestoon, "Craftsmanship over Crap". Tämän kaiken taustalla on, että ohjelmoijat ovat väsyneet kirjoittamaan surkeaa koodia ja haluavat sen sijaan tehdä hyvää jälkeä[3]. Pian sen jälkeen Software Craftsmanship -liike lähti liikkeelle perustamalla oman manifestonsa. Kokoontumispaikkana samanmielisille on toiminut SC-yhteisön postituslista.

Nousu

Keskeisessä osassa Software Craftsmanship -piireissä on oman osaamisensa jatkuva kehittäminen. SC-purjeen alla on järjestetty coding dojoja, esitetty code katoja, pidetty konferensseja ja kursseja, vaihdettu työntekijöitä firmojen välillä ja jotkut ovat lähteneet kiertueelle päästäkseen parikoodaamaan useiden ihmisten kanssa.

Yksi kirjaimellinen "journeyman" on Corey Haines, joka aloitti joulukuussa 2008 noin vuoden kestäneen kiertueen, jossa hän matkusti ympäri Pohjois-Amerikkaa ja pariohjelmoi ihmisten kanssa yösijan vastineeksi. Hän piti videoblogia matkoistaan ja sen jälkeen hän on ollut yksi SC-liikkeen keulahahmoista. Coreyn esitys "Software Craftsmanship, Beyond The Hype"[4]. QCon London 2010 -konferensista on hyvä tiivistelmä siitä, mistä SC:ssä on kyse.

Useissa paikoissa ympäri maailmaa on pidetty *coding dojo* ja *code retreat* -tapahtumia, missä ohjelmoijat kerääntyvät yhteen hiomaan taitojaan, tyypillisesti toteuttaen pieniä ohjelmointitehtäviä pariohjelmoiden ja pareja kierrättäen. Suosittuja ovat myös *code kata* -harjoitteet, joissa tarkoituksena on ohjelmointitehtävän toiston kautta hioa taitojaan ja iskostaa niitä lihasmuistiin asti.

Monet ovat videoineet harjoittelemiaan katoja ja laittaneet ne toisten nähtäville palautteen saamiseksi ja osaamisen levittämiseksi[5]. Olen myös itse julkaissut *Let's Code* -screencasteja[6], joista näkyy miten TDD, refaktorointi, evolutionary design ym. menetelmät toimivat epätriviaaleissa projekteissa käytännössä.

Ovatpa jopa muutamat firmat vaihtaneet työntekijöitä kilpailijoittensa kanssa viikon ajaksi, jotta he saisivat uusia vaikutteita. Toiset firmat ovat järjestäneet craftsmanship-henkisiä sisäisiä valmennusohjelmia – mm. BBC:llä on kokeiltu mielenkiintoista vertaisoppimismenetelmää[7].

Software Craftsmanship ei ole mikään uusia asia. Jason Gorman sanoikin osuvasti, että SC ei ole "seuraava iso asia", vaan yritys pukea sanoiksi mikä kyseinen "asia" on aina ollut. Aiemmat design patterns ja agile-liikkeet ovat keskittyneet parempiin tapoihin tehdä asioita, mutta craftsmanship-liike pyrkii kasvattamaan parempia tekijöitä. ■

Software Craftsmanship ei ole "seuraava iso asia". Se on yritys pukea sanoiksi mikä kyseinen "asia" on aina ollut.

- Jason Gorman

[http://twitter.com/#!/jasongorman/status/20455097721]

Lähteet

1 [http://pragprog.com/magazines/2011-02/agile--]

2 [http://blog.jaoo.dk/2008/06/19/is-it-professional-to-cheat-the-boss/]

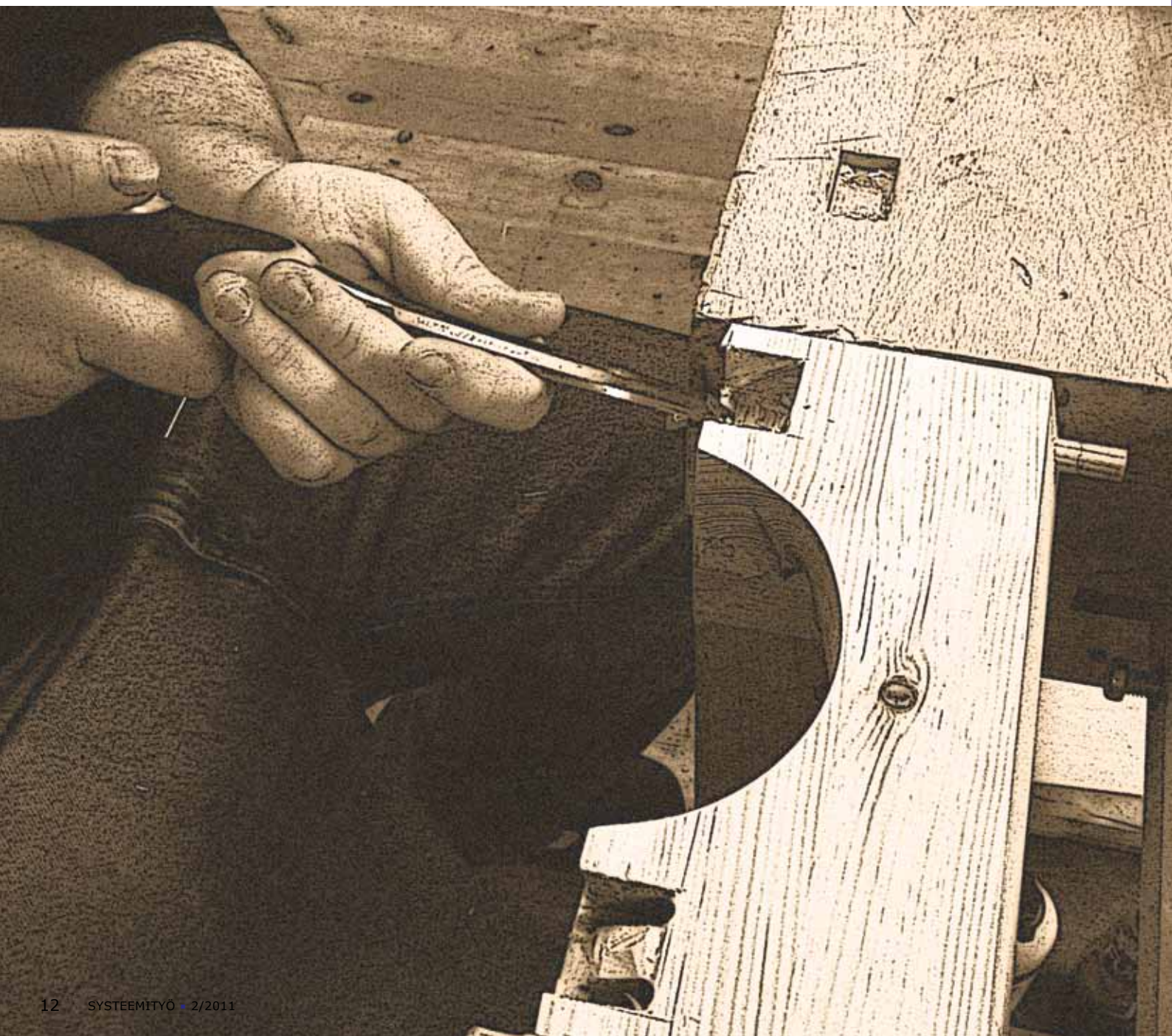
3 [http://cleancoder.posterous.com/software-craftsmanship-things-wars-commandmen]

4 [http://www.infoq.com/presentations/Software-Craftsmanship-Beyond-The-Hype]

5 [http://katas.softwarecraftsmanship.org/]

6 [http://www.orfjackal.net/lets-code]

7 [http://parlezuml.com/blog/?postid=984]



SYTYKE ry järjestää jo 14. laivaseminaarin 7. - 9.9.2011 (Helsinki-Tukholma-Helsinki):

Muurit murtuu! - Oletko valmis muutokseen?

Risteily on vuosittain saanut erinomaista palautetta tavasta, jolla IT-ammattilaisjoukko kerätään kuulemaan ja keskustelemaan seminaarin teemasta.

Sytyke ry:n tämän vuoden risteilyteemana on ”Muurit murtuu - Oletko valmis muutokseen?”, jossa keskitytään moneen eri murrokseen, joita parhaillaan koemme.

Tänä vuonna käytössämme on auditorio, johon mahtuu 180 henkeä ja tila voidaan jakaa kahtia. Tämä mahdollistaa osittain 2-linjaisen ohjelman - toisessa salissa käsitellään liiketoiminta-, toisessa teknistä näkökulmaa.

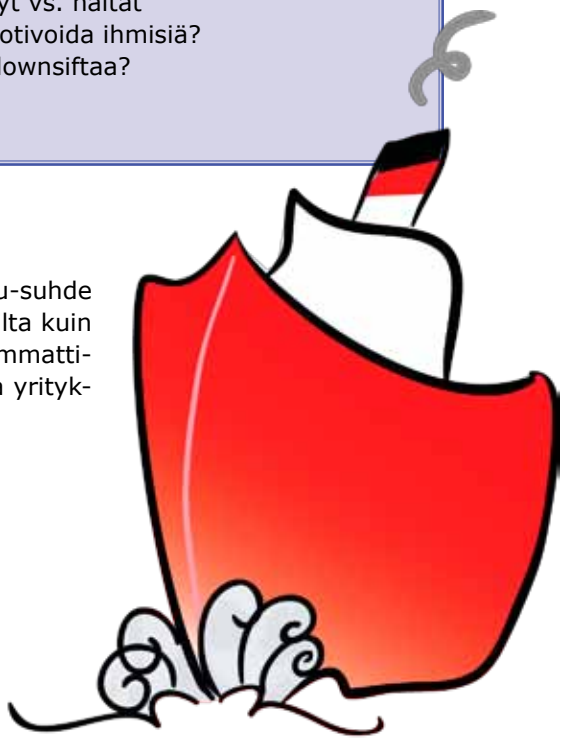
Teemaa käsitellään esimerkiksi seuraavista näkökulmista:

- Kuinka pilviteknologiat muuttavat IT-toteutuksia, hankintamalleja ja hankintasopimuksia
- Avoin data: Miksi yrityksen pitäisi antaa tietoja edelleenkäytettäväksi? Mitkä businessmallit, mistä raha?
- Toteutusalustojen valinnan vaikeus: Windows, Linux vai MacOS? Android, iOS, WP7 vai Meego? Mihin kannattaa panostaa?
- Teknologiamurros: ipv4 vs. ipv6, web 1.0/web 2.0/mobiiliappsit
- Viestintä: Virallisesta viestinnästä 2-suuntaiseen kommunikointiin? Ajasta ja paikasta riippumaton viestintä: Hyödyt vs. haitat
- HR: Työn ja vapaa-ajan raja hämärtyy. Millä motivoida ihmisiä? Miten pidetään hommat pyörimässä, kun osa downsiftaa?

Osallistujilta saadun palautteen perusteella seminaarien hinta-laatu-suhde on ollut erinomainen ja sen ohjelma on saanut kiitosta niin osallistujilta kuin sponsoreiltakin. Seminaariin osallistuu ohjelmistojen kehittämisen ammattilaista ja päättäjää sekä ohjelmistotaloista että ohjelmistoja tilaavista yrityksistä ja organisaatioista.

Lisätietoja löydät osoitteesta http://tinyurl.com/laivaseminaari2011

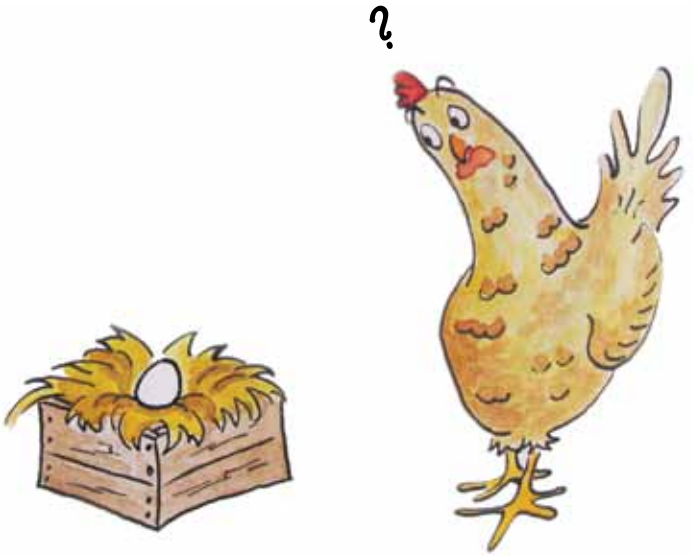
TERVETULOA OSALLISTUMAAN!



Raine Kauppinen
[raine.kauppinen@haaga-helia.fi]

Kirjoittajat ovat työskennelleet ohjelmistokehitystehtävissä, sekä julkisen että yksityisen puolen projekteissa. Tällä hetkellä kirjoittajat toimivat opettajina HAAGA-HELIA ammattikorkeakoulussa ensisijaisesti systeemityön ja ohjelmoinnin opetustehtävissä. Esitetyt mielipiteet ovat tekijöiden omia, eivätkä välttämättä edusta kirjoittajien työnantajan melipiteitä.

Altti Lagstedt
[altti.lagstedt@haaga-helia.fi]



Scrum käyttöön – Nollasta sataan 16 viikossa?

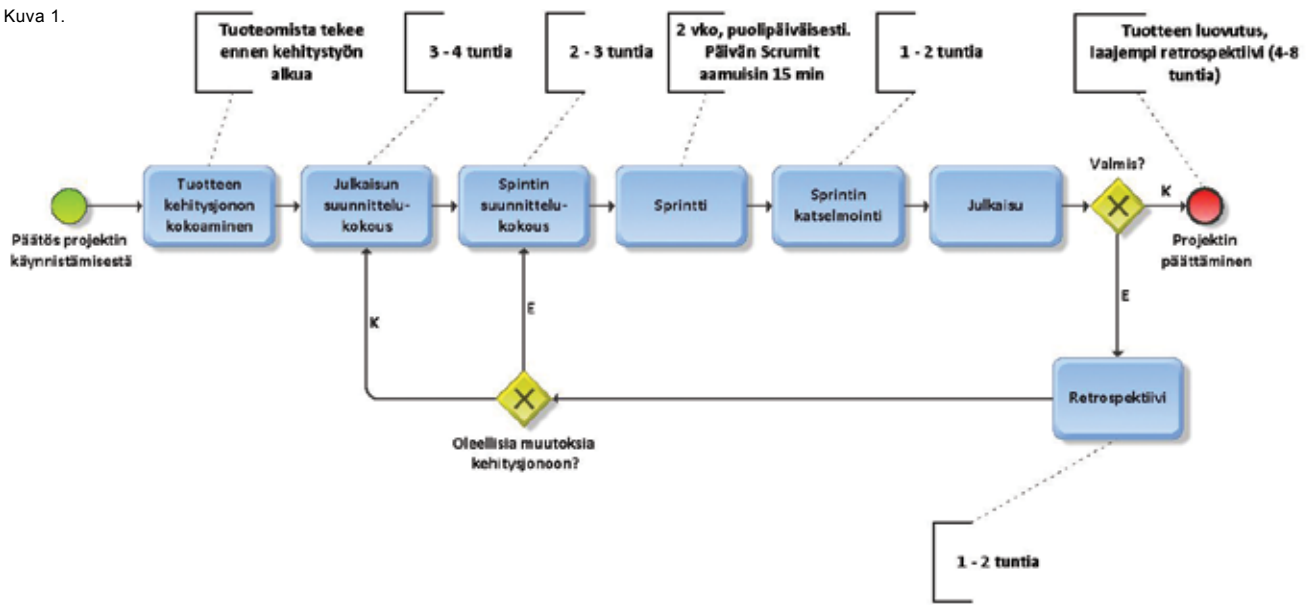
Pitääkö ohjelmistokehityksessä ensin tietää miten kohdealuetta ymmärretään, vai mikä on teknisesti mahdollista? Voiko kohdealueen tarpeita ja erityisesti niitä toteuttavia ratkaisuja oikeasti ymmärtää, jos ei ymmärrä tekniikasta mitään? Onko teknisestä ymmärryksestä mitään hyötyä, jos ei ymmärrä kohdealueesta mitään? Eteneekö työ tekniikan vai käyttötarpeen ehdoilla? Kumpi tulee ensin, muna vai kana? Vai tuleeko sittenkin ensin sika?

Mitä tehtiin?

Syksyllä 2010 aloitettiin HAAGA-HELIA ammattikorkeakoulun Tietotekniikan koulutusyksikössä uusi Scrum-pohjainen Ohjelmistokehitysopintojakso, jossa otettiin uusi lähestymistapa systeemityön käytäntöjen oppimiseen. Taustalla olivat aiemmat systeemityöopintojaksot, joista haluttiin mukaan ottaa onnistuneita osuuksia. Toisaalta haluttiin myös ottaa oppia aiempien toteutusten haasteista ja tuoda työelämään sijoittautumisen käytäntöjä mukaan opintoihin. Koska nyt toteutettu opintojakso oli aikaisempia selvästi pienempi, oli myös painotuksia pakko vaihtaa. Aiemmin ohjelmistokehityksen teorian, dokumentoinnin ja projektinhallinnan painoarvo oli suurempi, nyt päätettiin lähestyä kohdetta ketterästi tutkivan ja ongelmalähtöisen oppimisen periaatteilla (ks. esimerkiksi Hakkarainen, K. et al. 2005 ja Poikela, E. & Poikela, S. 2005). Tutkiva ja ongelmalähtöinen oppiminen muistuttaa hyvin paljon niitä käytäntöjä, joiden kautta työelämässä haetaan uutta tietämystä ja otetaan uusia menettelytapoja käyttöön. Opintojakson perustiedot ja tavoitteet tietonurkassa 1.

Kehittämisen lähtökohtana kehitystiimit saivat aiemmin toteutetun yksinkertaisen runkosovelluksen dokumentaatioineen. Runkosovelluksen ajateltiin olevan tuoteomistajan aiemmin, eri kehitystiimiltä tilatun ja perinteistä vesiputousmallimaista ohjelmistokehitystä noudattaneen kehitystyön tulos. Tätä runkosovellusta kehitystiimit alkoivat jatkokehittää ketterällä menetelmällä. Runkosovellus oli toteutettu Java EE -ympäristössä servlettejä ja jsp-sivuja käyttävänä tietokantapohjaisena websovelluksena. Sovellusta myös jatkokehitettiin tässä samassa ympäristössä. Opintojakson tavoitteena ohjelmointitekniikan kannalta olikin perehdyttää opiskelijat verkkosovellusten toteuttamisen tekniikoihin. Kehitysympäristönä oli eclipse, versionhallintana CVS, palvelimena Tomcat ja tietokantana MySQL. Tuotteen ja julkaisun kehitysjonot, sprittien tehtävälistat ja edistymiskäyrät olivat Excelissä. Opintojaksoa varten muodostettiin ns. ohkescrum-malli (kuva 1), jonka puitteissa kehitystyö tapahtui. Jokainen kehitystiimi teki omaa itsenäistä ratkaisuaan. Toisin sanoen, vaikka kaikki tiimit lähtivät samasta tilanteesta, niin kaikilla tiimeillä oli omat, erilliset tuotteen kehitysjonot.

Kuva 1.



Tuoteomistaja priorisoi eri tiimien käyttäjätarinat erilailla, mikä mahdollisti eri tiimien erilaisten tilanteiden huomioimisen.

Mitä jäi käteen?

Toteutuksellamme saimme kokemuksia kaikista oppimistavoitteina mainituista osa-alueista. Kattavamman systeemityömenetelmistön ja projektihallinnan osalta opiskelijoiden kokemus jäi pääosin johdantoviikon ryhmätöiden varaan. UML:n soveltamista tuli vastaan sekä runkosovelluksen dokumentaatioon tutustuttaessa että keskeisiä UML-kuvauksia päivittäessä. Merkittävimmät kokemukset liittyivät kuitenkin Scrumiin ohjelmistokehitysmenetelmänä, estimointiin ja ryhmätyöskentelyyn liittyen. Näitä kokemuksia on kuvattu seuraavaksi tarkemmin.

Scrumista ohjelmistokehitys- ja perehdyttämismenetelmänä

Opintojakson aikana opiskelijat toteuttivat olemassa olevaan sovellukseen uusia ominaisuuksia tuoteomistajalta ja järjestelmäasiantuntijalta saatujen käyttäjätarinoiden mukaan. Käyttäjätarinat olivat aluksi osin epäloogisessa järjestyksessä, joiltakin osin ristiriitaisia ja tuoteomistajan toiveet ja tavoitteet vaihtelivat työn edetessä. Kehitystiimikohtaisten tuotteiden kehitysjonojen havaittiin toimivan suunnitellulla tavalla ja tukevan tiimikohtaista etenemistä, sillä mikä oli yhdelle tiimille helppoa, saattoi toiselle tiimille aiheuttaa haasteita ja päinvastoin. Kehitysjonojen tukemalla tiimikohtaisella ohjauksella pystyttiin aidosti huomioimaan tiimien tilanne ilman että muiden tiimien tekemiset vaikuttivat etenemiseen.

Havaitsimme myös, että kehitysjonoja on mahdollista käyttää sekä tuoteomistajan (tilaajaorganisaation edustaja) että järjestelmäasiantuntijan (kehitysorganisaation edustaja) tavoitteiden kommunikoimiseen kehitystiimeille. Käyttäjätarina sisältää ensimmäisenä kohtanaan sen, kenen tarpeesta on kyse, tällöin kehitystiimeille käy selkeästi ilmi onko kyse tilaajan vai oman kehitysorganisaation tavoitteista. Oikeisiin työkäytäntöihin oppimisen ja perehtymisen kannalta kehitysjonoon voidaan siis laittaa myös tarinoita, jotka edistävät haluttua oppimista, eivätkä niinkään ole tarkoitettu ohjelmiston ominaisuuksien lisäämiseen tai parantamiseen. Tällaiset tarinat voivat ohjata esimerkiksi organisaatiossa käytetyn arkkitehtuuriin tutustumiseen tai uusien kehitysmenetelmien ja -tekniikoiden selvittämiseen sekä myös opittaessa organisaation työkäytäntöihin. Tämän käytännön hyvänä puolena on, että oppiminen tapahtuu samalla tavoin kuin varsinainen kehitystyökin.

Ketterässä ohjelmistokehityksessä kaaos on aina nurkan takana: Kun sovellusta ei etukäteen määritellä eikä suunnitella yksityiskohtaisesti, on vaikea luoda alusta asti toimivaa sovellus- ja tietoarkkitehtuuria. Opintojaksolla kehitystyötä tehtiin ketterällä periaatteella: Tee se ensin toimivaksi, sitten vasta nopeaksi. Vähitellen tuoteomistajan muuttuneiden tietotarpeiden vuoksi olemassa olevan ohjelmakoodin

TIETONURKKA 1

Ohjelmistokehitys –opintojakso 2010

Opiskelijoiden lähtötaso:
JAVA perusosaaminen, ei kokemusta Java EE ympäristöstä, eikä ryhmässä tapahtuvasta ohjelmistokehittämisestä

Toimijat ja heidän roolinsa:
2 opettajaa, toinen opettajista toimi tuoteomistajana ja toinen järjestelmäasiantuntijana sekä agile coach –roolissa
20 kolmannen lukuvuoden opiskelijaa, neljä viiden henkilön ryhmää

Oppilaiden eri roolit:
johdantoviikoilla ryhmätyöt, kehitystiimeissä ohjelmistokehitystyöntekijänä toimittajan roolissa
Aikaa 16 viikkoa, 18 työtuntia viikossa

Työaika ja toimintamalli:
Työajat olivat kiinteitä, ei ylitöitä, mutta ei myöskään poissaoloja sallittu. Kehitystiimin jäsenet työskentelivät kaikki samassa huoneessa.

Tavoite oli opiskelijoiden sisäänajaminen työelämän käytäntöihin:
Systeemi- ja projektityötä oppiminen
Scrumin käyttö yhtenä systeemityömenetelmänä,
Työmäärien arvioinnin (estimoinnin) oppiminen,
Ryhmätyöskentelytaitojen ja ryhmädynamiikan ymmärtäminen ja soveltaminen
UML:n soveltamista
Java EE ympäristön ymmärtäminen
Hyvien ohjelmointikäytäntöjen soveltaminen

rakenne alkoi rapautua ja estää toteutustyön jatkamisen: uudessa tilanteessa vanhalla ratkaisulla ei voitu enää edetä. Silloin oli hyvin luontevaa perehdyttää tiimit refaktorointiin. Samalla päästiin myös pohtimaan suunnittelumallien ja muiden hyvien käytäntöjen merkitystä tämänkaltaisessa työssä. Oppimisen kannalta käytännön tilanne avasi varmasti näiden asioiden merkityksen paremmin kuin perinteinen teorialuento. Refaktorointi ja suunnittelumallit tulivat esille oikeasti hyödyllisinä ja hyödynnettävinä työkaluina eivätkä teoreettisina pohdintoina.

Perinteisessä mallissa, jossa koodi tehdään tarkan suunnitelman pohjalta, ei tuollaista oppimistilannetta ole niin helposti järjestettävissä, sillä siihen ei välttämättä ole aikaa, kun työtä ohjataan helposti vain kohti ennalta suunniteltua tavoiteltua ratkaisua. Ketterässä ohjelmistokehittämisessä taas voidaan päätyä refaktorointia vaativaan tilanteeseen luontevasti, sillä se on osa ketterän kehittämisen perusideologiaa, eikä se tarkoita, että olisi vielä tehty suuria virheitä. Tässä mielessä ketterä ohjelmistokehitys tuki erinomaisesti opiskelua ja se mahdollistaa myös yrityksissä luontevan tavan perehtyä käytössä oleviin hyviin käytäntöihin. Opintojaksolla opitut hyvät käytännöt tietonurkassa 2.

Estimoinnista

Suunnitellessamme toteutustamme valitsimme yhdeksi toteutuksemme keskeisimmistä tavoitteista sen, että opiskelijat oppivat estimointia, eli työmäärien ja resurssitarpeiden arviointia. Työmääräarviointi on projektin onnistumisen kannalta tärkeää, joskin haastavaa ja sen oppiminen hidasta. Perinteisiä ohjelmistokehitysmenetelmiä käytettäessä kattava palaute tehdystä estimoinnista saadaan vasta projektin päätyttyä, jolloin estimoinnit on tehty kuukausia aiemmin, ja tällöin estimoinnin oppiminen kokemuksen kautta jää heikoksi. Perinteisissä projekteissa estimointitaidon kehittymistä vaikeuttavat lisäksi eri sisältöiset projektit (erilaiset tavoitteet, erilaiset aikataulut, eri asiakkaat) ja eri projektien erilaiset resurssoinnit (eri tekijät, eri määrä tekijöitä).

Näimme tässä selkeän kehityskohteen ja tilaisuuden, sillä Scrum tarjoaa mahdollisuuden nopeaan palautteeseen estimoinnin onnistumisesta: Jokaisen sprintin jälkeen tehdään katselmointi ja seuraavan sprintin suunnittelu. Myös sprintin aikana tehdään tarvittaessa uudellensuunnittelua. Estimointia siis tehdään jatkuvasti ja palaute estimoinnin onnistumisesta saadaan näkyviin lähes välittömästi. Estimointia myös helpottaa että käytettävissä olevat resurssit ovat kiinteät: jokaisessa sprintissä on sama aika ja samat tekijät. Tällainen jatkuva estimointi ja siitä nopeasti saatava palaute toimikin toteutuksellamme erinomaisesti estimoinnin opiskelumenetelmänä.

Käytimme estimointitapaa, joka noudattelee pääosin Mike Cohnin kirjassaan esittelemää menettelyä (Cohn, M. 2006). Estimoinnissa käytettiin tarinapisteitä (storypoints), eli toteutettavien käyttäjätarinoiden koko arvioitiin suhteessa muihin käyttäjätarinoihin. Vertailun pohjana käytettiin jotain jo toteutettua käyttäjätarinaa (verrokki), jonka koosta tiimi oli yhtä mieltä. Uusia tarinoita verrattiin verrokkiin, ja pohdittiin onko uusi tarina pienempi, yhtä suuri vai suurempi kuin verrokki. Käytännön estimointimenetelmänä käytettiin arviointipokeria (planning poker) avulla. Tavoitteena ei ollut saada 100% oikeita arvioita, vaan päästä riittävään tarkkuuteen. Jo muutaman sprintin jälkeen tiimien estimoinnit osuivat hyvin tarkasti oikeaan. Ryhmätyöskentelystä ja -dynamiikasta. Toinen hyvin tärkeä tavoite opintojaksolla oli antaa opiskelijoille mahdollisuus kokea ja ymmärtää ryhmän dynamiikkaa ja tiimin kehittymisen vaiheita. Ryhmätyötaidot ovat työelämässä tänä päivänä yhä tärkeämpiä ja

TIETONURKKA 2

Käytetyt scrum-käytäntöjä täydentävät yleiset käytännöt:

- versionhallinta (cvs)
- 5-taso arkkitehtuuri
- suunnittelumallit
- refaktorointi
- pariohjelmointi
- kattava kommentointi
- koodin yhteisomistajuus
- jatkuva testaus
- jatkuva integrointi
- jatkuva julkaisu

ohjelmistokehityksessä erityisen tärkeitä: yksinäisiä ratsastajia harvoin kaivataan projekteihin.

Aku Kopakkalan (Kopakkala, A. 2005) mukaan tiimin kehittymisessä on tietyt vaiheet, ja tukemalla tiimin toimintaa oikeissa kohdissa pyrittiin varmistamaan että kaikki ryhmät pystyisivät kehittymään tehokkaiksi tiimeiksi. Ryhmän toimintaa pyrittiin tehostamaan antamalla alkuvaiheessa valmiimpia ja jäsennellympiä, yksinkertaisia tehtäviä, jotka toteutettiin ohjaavien opettajien vahvassa ohjauksessa ja myöhemmin haastavampia, tiimin itsenäiseen ongelmanratkaisukykyyn ja yhteistyöhön nojaavia tehtäviä. Tällöin saatiin alkuvaiheelle vahva (ulkoinen) johtajuus ja ryhmän toiminnan kehittyessä pystyttiin siirtämään vastuuta enemmän tiimeille jolloin tiimin itsenäinen ongelmanratkaisukyky korostui. Tiimiytymistä tuettiin myös käyttämällä kiertävää pariohjelmointia: kaikki tekivät jossain välissä töitä kaikkien muiden parina ja näin saatiin vähennettyä klikkiytymisen riskiä. Pidimme tärkeänä sitä, että tiimin jäsenet tiedostavat (ja mahdollisesti myös tunnistavat) tiimin kehitysvaiheet, jotta tiimi osaisi reagoida erilaisiin tilanteisiin ja pyrkiä kohti tehokasta toimintaa.

Ketterä kehittäminen ja erityisesti Scrum tarjoavat hyvän kehikon tiimitaitojen opettamiselle. Tiimille saadaan itsenäistä vastuuta nopeasti, käytännön toteutuksen konkretisointi ja organisointi jää nimenomaan tiimien vastuulle. Tiimien vastuuttaminen liittyy vahvasti scrumiin:
"Scrumtiimi koostuu scrummasterista, tuoteomistajasta ja kehitystiimistä. Scrumtiimin jäseniä kutsutaan "sioiksi". Tuoteomistaja on tuotteen kehitysjonon "sika". Kehitystiimi on sprintin tehtävälistan "sika". Scrummaster on Scrum-prosessin "sika". Kaikki muut ovat "kanoja". Kanojen ei tule kertoa sioille, kuinka heidän työnsä tulisi tehdä."
(Schweber, K. & Sutherland, J. 2010)

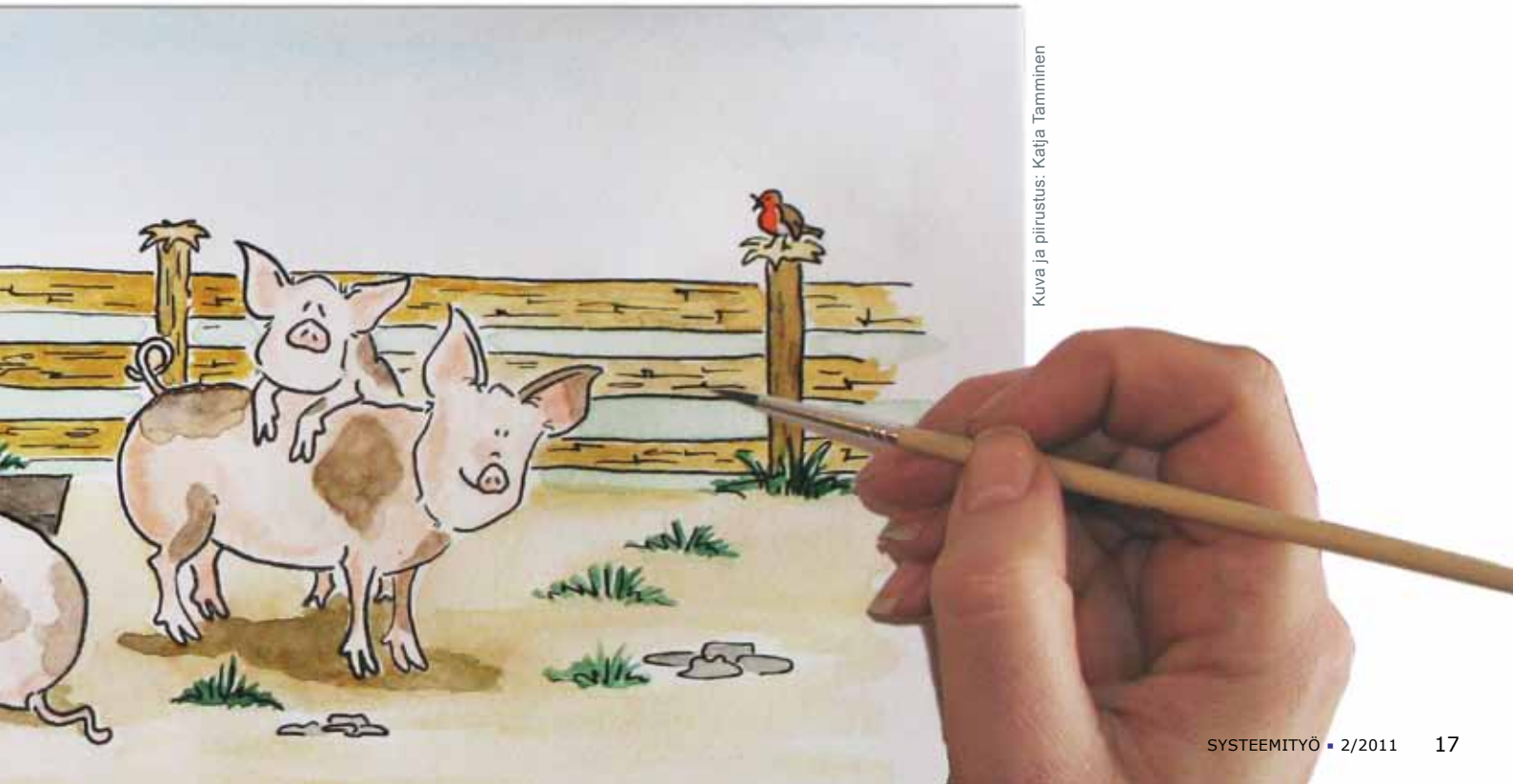
Toisaalta tiimejä ei myöskään jätetä oman onnensa nojaan: tuoteomistaja ja järjestelmäasiantuntija (tarvittaessa myös agile coach -roolissa) olivat koko ajan tiimien käytettävissä ja seurasivat työn edistymistä niin päivän Scrumeissa kuin muulloinkin. Päivän Scrumit myös edistivät kommunikaatiota tiimin sisällä, sillä niissä oleelliset asiat on pakko käydä yhdessä läpi eikä mitään tai kukaan ei jää pimentoon pitkäksi aikaa.

Kaiken kaikkiaan voidaan sanoa että tiimien kehittyminen oli selvästi havaittavaa ja pahimmat Kopakkalan mainitsemat riskit onnistuttiin välttämään. Käytetty ajanjakso oli melko lyhyt ja tiimien kehittymisen kannalta olisikin ollut hyödyllistä, jos opintojakso olisi jatkunut vielä pari kuukautta pidempään tai jos sitä olisi heti seurannut uusi opintojakso tai projekti samoilla kehitystiimeillä. Tiimien kehittymistä tukivat myös sprinttien retrospektiivit. Lisäksi saimme opintojakson lopulla Sytyke ry:n kautta Rami Talmen RP5 Oy:stä käymään laajempaa retrospektiiviä opiskelijoiden kanssa. Tämä kiinnitti opintojaksoa hyvin työelämään, kun Rami kykeni kertomaan työelämässä kerättyjä käytännön kokemuksia siitä, millaista Scrum-kehittäminen on ja mitä hyötyjä ja haasteita siihen liittyy. Oli myös hienoa huomata, että kehitystiimien kokemukset vastasivat erittäin hyvin Ramin esille tuomia asioita.

Mitä opimme?

Kaiken kaikkiaan toteutuksella saavutut tulokset ylittivät odotukset. Runsas ohjelmointi paransi opiskelijoiden ohjelmointitaitoa ja ymmärrystä Java EE -ohjelmoinnista oleellisesti, mutta hyvin oleellista oli myös sprinttien myötä kehittynyt taito estimoida käyttäjätarinoiden suuruuksia, kyky esittää hyviä kysymyksiä tuoteomistajalle, taito jakaa tehtäviä tiimin kesken ja kyky ymmärtää ohjelmistokehitystä kokonaisuutena. Varsinkin tuo toteutuksella nopeasti kehittynyt taito estimoida osoitti tämän tyyppisen oppimisen potentiaalisen vahvuuden.

Usein painotetaan, että ketterät menetelmät soveltuvat nimenomaisesti kokeneille ohjelmistokehittäjille, jotka tietävät miten "sääntöjä rikotaan hallitusti". Tällöin lähtöajatuksena on että pitää ensin opiskella huolella perinteiset systeemityömallit ja vasta sen jälkeen voi vasta siirtyä tekemään asioita ketterästi. Toteutuksella saamiemme


Kuva ja piirustus: Katja Tamminen

kokemusten perusteella asian voi mielestämme nähdä ja tehdä toisinkin: Voi lähteä liikkeelle hyvin pienestä, edetä ketterästi ja koko ajan kasvattaen lisätä tietämystä ja samalla tehdä tehokasta kehitystyötä. Kun erilaisiin ongelmiin törmätään, niihin reagoidaan välittömästi, vastaavasti kuin työelämässäkin on pakko tehdä.

Ei siis ole välttämätöntä käydä ensin teoriaa kattavasti läpi ja vasta sitten siirtyä soveltamaan sitä. Toteutuksellamme toimimme itse asiassa juuri päinvastoin: Ensin tehdään ja kun (aika piankin) törmätään ongelmiin, niin pidetään kyseiseen ongelmaan liittyen tietoisku tai opiskelijoille annetaan sopiva selvitystehtävä esimerkiksi käyttäjätarinana tuotteen kehitysjonossa. Tällainen tutkiva ja ongelmalähtöinen opiskelu on hyvin työelämässä toimimisen kaltaista. Lisäksi, kun tietoiskut ja selvitykset kohdistuvat opiskelijoiden kokemiin aitoihin ongelmiin, on kuulijakunta yleensä huomattavasti vastaanottavaisempaa kuin perinteisissä oppimistilanteissa, kuten esimerkiksi luennolla.

Mitä haluamme sanoa?

Pyrimme toteutuksellamme luomaan työelämän kaltaisen tilanteen ohjelmistokehitykselle. Opiskelijat laitettiin toimimaan kehitystiimeissä rakenteessa, jossa tilaaja edusti tuoteomistajana toiminut opettaja ja kehitysorganisaatiota edusti järjestelmäasiantuntijana ja tarvittaessa agile coach –roolissa toiminut opettaja. Myös ohjelmistokehityksen tavoite oli työelämässä usein vastaan tuleva: Jatkokehityksen kohteena on olemassa oleva, muiden alun perin kehittämä sovellus, jonka toteutuksessa käytetyt menetelmät ovat kokonaan tai osittain uusia kehitystiimille. Sovelluksen jatkokehitys tapahtui ketterällä menetelmällä kiinteässä julkaisuaikataulussa.

Opintojakson pohjana oli kirjallisuudesta saatu tietämys ketteristä menetelmistä, yritysmaailman (ja Sytyke ry:n) edustajien kanssa käydyt keskustelut sekä ohjaavien opettajien omat, aiemmista työtehtävistä kertyneet kokemukset ohjelmistokehityksestä. Taustalla olivat myös ohjaavien opettajien kokemukset aikaisemmista tavoista opettaa systeemityötä sekä tieto siitä, millaisia tapoja muualla vastaavassa opetuksessa käytetään.

Laajempi soveltaminen

Vaikka tässä artikkelissa on käsitelty uuden systeemityömallin opettamista ja oppimista oppilaitosympäristössä, monet havainnot ovat sovellettavissa hyvin myös työelämän vastaavissa projekteissa, varsinkin uuden tiimin perehdyttämisessä tai otettaessa Scrum tai muu ketterä kehitystapa käyttöön. Vaikka ketterää kehitystä ei yleensä suositella aloittelijoille, niin näkemyksemme on, että ketterä kehittäminen on toimiva tapa ajaa sisään aika aloittelevakin kehitystiimi, sillä ketterä kehittäminen mahdollistaa ”elefantin syömisen paloina”, myös kehitysorganisaation ohjelmointikäytäntöjen (esim. arkkitehtuurit, suunnittelumallit, kehitystyö- dokumentointikäytännöt) ja tiimityötapojen opiskelussa. Tämä edellyttää, että aloittelevaa tiimiä tukevat ja ohjaavat asiantuntijat, jotka ovat suunnitelleet edistymisaskeleet kehittäjäyhteisön tavoitteiden ja käytäntöjen mukaisiksi. Siirtyminen ketterään kehitykseen ei ole mutkatonta, mutta ottamalla askeleet oikeassa järjestyksessä pahimmilta karikoilta on mahdollisuus välttyä.

Miten tästä eteenpäin?

Saatujen kokemusten perusteella olemme opintojakson uudessa toteutuksessa (kevät 2011) luopuneet johdantoviikoilla käytetyistä teoria-aluennoista käytännössä kokonaan. Sen sijaan aloitimme sprintit heti opintojakson alusta, jolloin kaikki oppiminen tapahtuu Scrum-työmenetelmällä kehitysjonoissa esiintyvien selvitystehtävien ja niiden tuloksia käsittelevien tietoiskujen avulla. Tämä edellyttää opettajilta alusta lähtien vahvaa läsnäoloa kehitystiimien työskentelyssä ja entistä valmentavampaa ja konsultoivampaa otetta opetukseen: Opettajien on oltava työssä mukana siinä missä opiskelijoidenkin, perinteinen opettajan rooli on syytä unohtaa.

Myöhemmillä lukukausilla tapahtuvan oppimisen pitää suuntautua koko ajan enemmän kohti työelämää, jotta valmistuvan opiskelijan osaamisten ja työelämän todellisuuden välillä ei olisi kuilua. Tätä tavoitetta varten tarvitsemme työelämästä yhteistyökumppaneita, jotka ovat kiinnostuneita antamaan todellisia aiheita opiskelijoiden 4.-7. lukukauden ohjelmistoprojekteille ja toimimaan tilaajan ja mahdollisesti tuoteomistajan roolissa näissä projekteissa.

Toinen työelämäkumppaneita kaipaava kehityssuuntamme on kehittää ”ohkecrumista” työelämän perehdyttämiseen ja ohjelmistokehitysmenetelmien sisäänajoon soveltuva malli. Tämä työ on aivan alkuvaiheessa, joten tämän artikkelin perusteella heränneet tai muutoin aiheeseen liittyvät kommentit ja yhteistyöehdotukset ovat tervetulleita. ■

Lähteitä

Agilemanifesto.org 2001. Manifesto for Agile Software Development. Luettavissa (tarkistettu 27.2.2011): http://agilemanifesto.org/

Cohn, M. 2006. Agile Estimating and Planning. Pearson Education

Hakkarainen, K., Bollström-Huttunen, M., Pyysalo, R., & Lonka, K. 2005. Tutkiva oppiminen käytännössä: matkaopas opettajille. Helsinki, WSOY

Kauppinen, R. & Lagstedt, A. (2011). Progressive inquiry in agile software development education. Interdisciplinary Studies Journal, Special Issue: Encounters11 - Developing Competences for Next Generation Service Sectors, Vol. 1, No. 3. To appear on June 2011

Kopakkala, A. 2005. Porukka, jengi ja tiimi. Edita, Helsinki

Poikela, E. & Poikela, S. 2005. PBL - Ongelmista oppimisen iloa. Tampere, Vammalan Kirjapaino Oy

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. Prentice Hall

Schwaber, K. & Sutherland, J. 2010. Scrum. Luettavissa (tarkistettu 27.2.2011): http://www.scrum.org/scrumguides/

Rami Talme on RP5 Softwaren toimitusjohtaja ja konsultti joka toimii Agile Coachina. Rami on mukana myös Sytykkeen hallituksessa.
[rami.talme@rp5.fi]

Turku Agile Day011

Täällä sitä taas ollaan Sytykkeen kirjeenvaihtajan ominaisuudessa. Viime vuonna aloitettu TurkuAgileDay -tapahtuman raportointi sai sen verran hyvää kaikua, että se päätettiin uusia.

Tänä vuonna pääsin osallistumaan ainoastaan varsinaiseen seminaaripäivään joten workshopit jäivät valitettavasti väliin. Luotettavalta taholta kuultuna voidaan kuitenkin todeta, että workshopit olivat kuulemma jopa parempia kuin viime vuonna. Ja olivat taas oikeasti sellaisia, että siellä olleet ottivat oikeasti osaa tekemiseen.

Ensimmäisen keynote:n aihe (Pierluigi Pugliese & Mike Sutton) vaikutti lupaavalta: ”Soft Skills Essentials for Software Craftsmen”. Kaverit olivat hieman tehneet tutkivaa journalismia ja etsineet Agile -kirjallisuudesta sanaa ‘person’. Tulokset olivat aika laihoja. Tiimi näkyi kyllä aika vahvana mutta yksilöt olivat hukkuneet tiimien sekaan. Ulkomaan poijat ottivat vahvasti kantaa muistuttamalla, että ”Teams are made of individuals”. Pierluigi ja Mike heittivät ilmoille teesinsä jolla yksilöä voidaan paremmin tukea jotta hän samalla toimisi tiimissä paremmin ja tehokkaammin.

Tiimi näkyi kyllä aika vahvana mutta yksilöt olivat hukkuneet tiimien sekaan

Yksilölle:
1) pitää antaa mahdollisuus oppia uutta
2) tarjota mahdollisuus hyödyntää kokenutta mentoria
3) tarjota mahdollisuus kommunikoida
4) tarjota yksilöllistä opastusta

”Anna niin saat” on malli jonka kaikki voisivat varmasti ottaa itselleen toimintatavaksi.

Seuraavaksi olikin mielenkiintoinen esiintyjä jota itse odotin kovasti. Martin von Weissenberg Nokialta tuli kertomaan millainen on ”The Bastard Agile Coach From Hell”. Ei siksi, että sellainen kokisin olevani tai haluaisin olla. Erittäin provokatiivinen otsikko oli kooste omakohtaisista kokemuksista ja listaus Lesson Learned -asioita. Martinilta irtosi päivän paras lainaus joka oli peräisin hänen kollegaltaan liittyen Meegoon. ”Kyllä me tää sössitiin ihan virallisesti”. Kaverihan kuulosti ihan rehelliseltä tuon aloituksen jälkeen. Hän kertoi myös loistavan esimerkin kuinka Peer Pressure ilmeni heidän projektissa. Kun oli sovittu, että myöhässä saapuminen oli sanktioitu euron laittamisella viihdekassaan ja eräs tiimin jäsen oli aina myöhässä niin mitä tapahtuikaan. Tämä kaveri halusi maksaa etukäteen pari kymppiä kerralla kun hän aina tulee kuitenkin myöhässä. Mutta tiimi alkoikin antaa kaverille vaihtorahaa parista kympistä jotta häneltä veloitettaisiin se tiimin sopima euro. Tällä tavalla voitiin hienosti ilmoittaa tiimin tinkimättömyydestä sopimuksiinsa. Tästä syntyi päivän paras vertaus ”This is our swimming pool, not your toilet”.

”Kyllä me tää sössitiin ihan virallisesti”

Kokemus oli jälleen kaikenkaikkiaan laadukas. Hyviä esityksiä ja kokemuksia ja taas tuli sellainen olo, että muutamia pieniä asioita sai poimittua joita on tullut heti koitettua käytännössä. Suosittelen lämpimästi ensi vuonna osallistumista vaikka tai juuri siksi, että tapahtuma Turussa on.

Ketterin terveisin,
Rami

PS. Yksi asia minkä olin unohtanut viime vuodesta muistui mieleen heti kahden minuutin päästä ensimmäisen keynoten alusta. Täällä osallistetaan seminaarikävijät eli heti piti tehdä muutaman minuutin harjoitteita vierustoverin kanssa. Ja se tuntui onnistuvan jälleen kotimaisen ‘jäyhän’ suomalaisen joukonkin voimin.

Täällä osallistetaan seminaarikävijät

”This is our swimming pool, not your toilet”

Peter Hägg
Kirjoittaja on toiminut ohjelmistoalalla yli 10 vuotta, josta suurimman osan konsulttina erilaisissa ohjelmistokehityksen rooleissa. Hänellä on myös yli viiden vuoden kokemus esimiestehtävistä. Viime vuodet hän on toiminut menetelmäkonsulttina.
[peter.hagg@codebakers.fi.]

Mestari-kisälli -ajattelu nykyaikaisessa ohjelmistotyössä

Käsityöläisyyden mestari-kisälli –ajattelu juontaa keskiajan Eurooppalaisesta perinteestä, jonka tarkoituksena oli varmistaa osaamisen siirtyminen sukupolvelta toiselle. Tässä artikkelissa pohditaan miten mestari-kisälli –ajattelua voisi soveltaa nykyaikaisessa ohjelmistokehityksessä. Lisäksi artikkelissa tarkastellaan onnistuneen mestari-kisälli –ajattelun soveltamista mestarin, kisällin ja esimiehen näkökulmasta.

Käsityöläisyyden historia

Käsityöläisyys (craftsmanship) oli järjestelmä, jonka juuret ovat keskiajan Euroopassa. Järjestelmän tarkoituksena oli kouluttaa uusi sukupolvi jonkin käsityötaidon osaajiksi. Järjestelmässä käsityöläiset, kuten sepät ja puusepät, jaettiin yleensä kolmeen eri taitotasoon. Taitotasot olivat: oppipoika (apprentice), kisälli (journeyman) ja mestari (master craftsman).

Oppipojat aloittivat yleensä 10-15 vuotiaina, jolloin he siirtyivät äidin helmoista mestarin suojellukseen. Mestari tarjosi oppipojille omien taitojensa lisäksi majapaikan jossa asua, ruokaa ja töitä. Vastineeksi tälle, oppipojat suorittivat yksinkertaisia tehtäviä ja oppivat tätä kautta alan perustaidot. Oppipoikana oltiin yleensä seitsemän vuoden ajan, jonka jälkeen oppipojasta tuli kisälli.

Kisällin päätehtävänä oli syventää ja soveltaa laaja-alaisesti oppipoikana opittuja tietoja ja taitoja. Kisällin toissijaisena tehtävänä oli opettaa uusille oppipojille samaiset perustiedot ja -taidot. Kisällillä oli kaksi periaatteellista uravaihtoehtoa, joko hän jatkoi matkustavana kisällinä loppuikänsä tai sitten hän eteni mestariksi. Matkustavat kisällit siirtyivät kylästä toiseen, samalla oppien ja jakaen uutta tietoa ja taitoja kylien välillä, mestarilta toiselle.

Kisällistä siirryttiin mestariksi siinä vaiheessa, kun siihen oltiin valmiita. Panostus siirtyä kisällistä mestariksi oli suuri. Ensin piti tehdä eräänlainen lopputyö (masterpiece), joka toimi osaamisen todisteena. Tämän lisäksi piti maksaa eräänlainen sertifiointimaksu killalle. Kolmantena asiana piti vielä kerätä varat oman työpajan perustamiselle. Kuten arvata saattaa, olivat panostukset sitä luokkaa, että useat kisällit pysyivät matkustelevina kisälleinä loppuikänsä. Mestareita oli täten aika rajoitetusti, esim. yksi puuseppä kylää kohti.

Järjestelmänä käsityöläisyys poistui teollisen aikakauden alussa. Tänä päivänä järjestelmä kiltoineen olisi ainakin joltain osin laiton, mutta sen vaikutus näkyy yhä esim. suomalaisessa koulutusjärjestelmässä.

Käsityöläisyys oppimisen näkökulmasta

Järjestelmästä voidaan kuitenkin ottaa vaikutteita ja soveltaa niitä ohjelmistotyöhön liittyvän osaamisen kehittämisessä. Oppipojat, kisällit ja mestarit toimivat tiiminä, yhteistyössä, tarkoituksenaan siirtää osaamista sukupolvelta toiselle, varmistaen käsityöläisyysperinteen säilyminen. Oppipoikien ja kisällien taitojen kertyessä, myös lopputulosten laatu parani. Mestari tuki osaamisen kehittämistä jakamalla omia kokemuksiaan, sekä vastavuoroisesti oppimalla uusia asioita oppipojilta ja kisälleiltä.

Kun asiaa tarkastellaan 70/20/10 oppimissäännön näkökulmasta - jossa 70 % oppimisesta tapahtuu työtä tekemällä, 20 % saadun palautteen perusteella tai seuraamalla muita ja 10 % formaalin koulutuksen kautta - niin ei ole yhtään ihmeellistä, että järjestelmänä käsityöläisyys toimi usean sadan vuoden ajan.

Välillä tuntuu, että nykypäivän hektisessä ohjelmistotyössä, jossa tärkeimpinä asioina ovat tehokkuus ja suorittaminen, ovat vanhat opit päässeet unohtumaan. Alalle koulutetaan jatkuvasti uusia henkilöitä ja teknologiat kehittyvät kiihtyvää tahtia. Ohjelmistokehitystyöhön liittyvät perusasiat eivät kuitenkaan ole juurikaan muuttuneet tänä aikana. Samalla kuitenkin alalla 10-30 vuotta olleet ammattilaiset joko siirtyvät myynti-, projektinhallinta- ja esimiestehtäviin tai saavat pahimmassa tapauksessa kenkää. Eikö olisi korkea aika käyttää hyväksi mestareiden kokemus, sen sijaan, että sitä väheksytään?

Käsityöläisyyden soveltaminen

Käsityöläisyyttä voidaan soveltaa ohjelmistotyössä monella eri tavalla. Olen valinnut mentoroinnin yleisenä soveltamistapana, koska se muistuttaa hyvin läheisesti käsityöläisyyden perusajatusta. Mentorointi vaatii mentorilta ja mentoroitavalta paljon panostusta sekä lisäksi tiettyjä ominaisuuksia ja kykyjä. Tarkastelen seuraavaksi käsityöläisyyden soveltamista kolmesta eri näkökulmasta, eli mentoroitavan, mentorin sekä esimiehen näkökulmasta.

Oppipojat, kisällit ja mestarit toimivat tiiminä, yhteistyössä, tarkoituksenaan siirtää osaamista sukupolvelta toiselle.

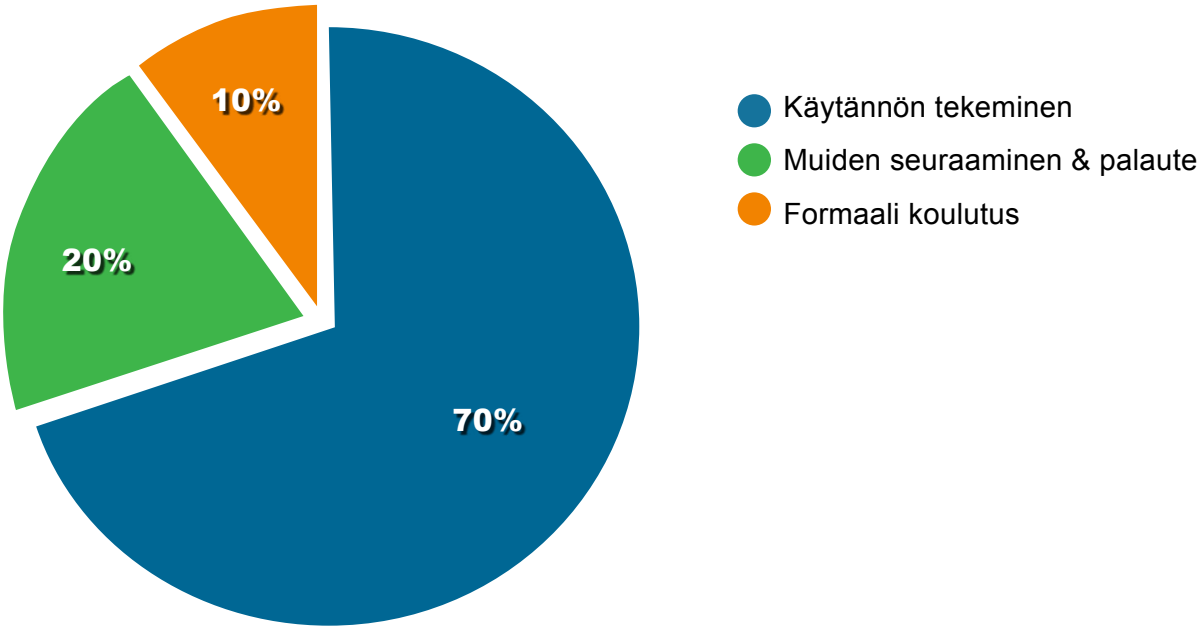
Ohjelmistotuotannon näkökulmasta (software craftsmanship) tulisi huomioida tietyt ajatukselliset erot alkuperäiseen käsityöläisyyteen. Ensinnäkin, ohjelmistotuotannon käsityöläisyyden äänekkäimmät edustajat eivät juurikaan kannata sertifiointeja (ja niihin liittyviä maksuja). Sertifiointi kun on harvoin tae siitä, että henkilö on jonkin aihe-alueen asiantuntija. Toisekseen, käsityöläisyyden harjoittaminen on tehty nykypäivänä helpommaksi. Yleisesti ottaen riittää, että kisälli kustantaa itselleen tarvittavan peruskoulutuksen. Valveutuneilla työnanatajilla on yleensä omat osaamisen kehittämiseen tähtäävät ohjelmat ja työpajat, joilla voidaan tukea työntekijän omaa kehittymistä.

Mentoroitavalle

Kaikki lähtee liikkeelle siitä, että mentoroitavalla tulee olla halu ja innostus kehittää avoimesti omaa osaamistaan ja taitojaan. Mentoroitavalla olisi hyvä omata kyky jatkuvasti mukautua ja muuttua ympäristöltä saadun palautteen pohjalta. Tähän liittyy myös kyky tunnistaa omat heikkoutensa ja löytää näihin erilaisia käytännönläheisiä ratkaisuja.

Mentotoitavan tulisi olla valmis kokeilemaan uusia asioita, pelkäämättä epäonnistumista. Uusia asioita kokeillaan ja välillä epäonnistutaan. Tehdyt virheet ja kokemukset otetaan huomioon seuraavan kerran kun kokeillaan jotain uutta. Perimmäisenä ajatuksena tässä on se, että epäonnistumisista opitaan yleensä paremmin kuin onnistumisista.

Mentoroitavan tulee olla valmis ottamaan vastuu omasta kehittymisestään. Tosiasia on, että mentori ei ole kaikkitietävä, vaan pitää myös olla uskallusta olla oma-aloitteinen. Mentorin tehtävänä ei ole olla kertomassa jokaisessa tienhaarassa, mitä seuraavaksi tulisi tapahtua. On myös hyvä muistaa, että mentoreille tulee joskus hetkiä, jolloin he eivät osaa selittää, miksi jokin asia toimii. Tällöin tehdään asiat mentorin intuition pohjalta.



Viimeisenä asiana, muttei vähäisempänä: käsityöläisyys ei ole mikään uskonto. Hyväksi todettuja asioita ei tulisi hylätä vain sen takia, että käsityöläisyys olisi ainoa oikea tapa. Toimivassa oppimisjärjestelmässä hyödynnetään toimivaksi todettuja asioita, oli niiden tausta mikä tahansa.

Mentorille

Nopea ja oikea-aikainen palaute on yksi mentoroinnin tärkeimmistä asioista. Kisälliä olisi hyvä seurata jatkuvasti, sekä antaa palautetta mahdollisimman usein. Ei laivaakaan ohjata asettamalla kurssi ja menemällä tämän jälkeen nukkumaan.

Odotusarvot tulisi asettaa varhaisessa vaiheessa. Mentorin olisi hyvä määritellä yhdessä kisällin kanssa mitä asioita formaalissa mielessä tullaan käsittelemään, jottei tule pettymyksiä, suuntaan tai toiseen.

Kokeneet ihmiset olettavat usein, että tieto on itsestään selvä asia. Mentorin olisi hyvä varmistaa varsinkin mentorointisuhteen alussa, että kisälli ymmärtää ainakin perusteet käsiteltävästä aihealueesta, ennen aiheeseen syventymistä. Pidemmän päälle tästä tulee tapa, jolloin tähän ei tarvitse enää niinkään panostaa.

Tietoa ja kokemuksia tulee jakaa avoimesti. Asiat tulisi selittää niin yksinkertaisesti kuin mahdollista. Tämä on harvinainen taito ja mentoroitava tulee arvostamaan tätä varmasti.

Mentorin tulisi kuunnella, selittää ja opastaa, muttei ohjata. Mentori ei käske mentoroitavaa tekemään asioita, vaan selittää, auttaa ja opastaa. Mentoroitavan tulisi tehdä omat päätöksensä siitä, miten hän soveltaa saatua tietoa. Tavoitteita tulee asettaa sekä suunnitella miten ne saavutetaan. Muuten mentorointisuhde ajautuu jossain vaiheessa tuuliajomoodiin. Hyvä tapa on asettaa selkeitä välitavoitteita, jotta päätason tavoitteet voitaisiin saavuttaa mahdollisimman vähäisellä vaivalla. Mentorin olisi hyvä olla askel edellä mentoroitavaa, koska mentoroitava ei välttämättä tiedä, mihin hänen seuraavaksi tulisi panostaa. Hyvä mentori ei odota, että kisälli kysyy, vaan tuo hyvissä ajoin esille mahdollisia vaihtoehtoja.

Mentorin tulisi pyrkiä tilaan, jossa mentoroitava imee tietoa, sen sijaan, että sitä annetaan. Kun mentori on tilanteessa, jossa mentoroitavalle ei enää tarvitse osoittaa suuntaa tai mentoroitava löytää ja soveltaa tietoa omaaloitteisesti, on mentorointisuhteessa pitkälti saavutettu kaikki, mitä formaalilta mentorointisuhteelta voi odottaa. Työ on tehty.

Esimiehelle

Mestari-kisälli -ajattelu tuo tiettyjä erityispiirteitä johtamiseen. Jos asiaa tarkastellaan lähimmän esimiehen näkökulmasta, nousee esille oikeastaan samat asiat, kuin asiantuntijoiden johtamisessa.

Johtamistyylin osalta tärkein asia on sisäistää, että alaiset ovat asiantuntijoita ja että heille maksetaan siitä, että he ajattelevat. Esimiehen tehtäväksi nouseekin toimia koordinoijan roolissa, varmistaen, että alaiset saavat kaiken tarvitsemansa tuen, jotta asiantuntijoiden ja muun organisaation työnteko olisi mahdollisimman tehokasta. Tärkeimmiksi esimiestaidoiksi nousevatkin valmentamis- ja fasilitointitaidot.

Eikö olisi korkea aika käyttää hyväksi alalla olevien mestareiden kokemus, sen sijaan, että sitä väheksytään?

Käsityöläisyyttä soveltavassa organisaatiossa nostetaan ihmiset keskipisteeseen. Jos asiantuntijatiimi toimii tehokkaasti, jatkuvasti kehittäen omaa osaamistaan ja taitojaan, ovat lopputuloksetkin hyviä. Esimiehen tulee muistaa, että esim. projektissa työskentelevät ihmiset eivät ole suoraan korvattavissa toisilla ihmisillä. Tämä ajattelutapa saattaa toimia teollisuudessa liukuhihnan äärellä, muttei alalla, jossa ajattelu ja ongelmanratkaisutaidot ovat äärimmäisen tärkeässä asemassa.

Entä sitten?

Onko mestari-kisälli -ajattelu hyvä tapa kehittää osaamista sekä henkilökohtaisella, että organisaation tasolla? Käsityöläisyyden soveltaminen on tietyssä mielessä kallista. Se vie paljon aikaa ja se vaatii sitoutumista, kun enää ei olla koko ajan suorittamassa. 70/20/10 oppimissäännön näkökulmasta ollaan kuitenkin 90 %:n paremmalla puolella, jossa opitaan tiimissä työtä tekemällä. Oikein sovellettuna käsityöläisyysajattelu mahdollistaa kannustavan ilmapiirin, jossa ihmisillä on mukava olla. Samalla voidaan mahdollisesti välttää tilanteet, jossa vanhemmat asiantuntijat ajautuvat organisaation ulkopuolelle.

Mitä tapahtuu huomenna?

Jos olet sitä mieltä, että mestari-kisälli -ajattelu voisi olla kokeilemisen arvoi-

nen asia, niin kannattaa paneutua tämän artikkelin taustamateriaaliin tarkemmin. Nämä ovat:

Software Craftsmanship: The New Imperative, Pete McBreen, Addison-Wesley Professional, 2001. Ohjelmistoalan käsityöläisyysajattelun perusteos, joka toimii hyvänä silmien avaajana.

Apprenticeship Patterns, Dave Hoover, Adewale Oshineye, O'Reilly Media, Inc. 2009. Kirja, joka on täynnä hyviä käytännön vinkkejä siitä, miten toimia kisällinä.

"The Secret of Being a Great Mentor", http://www.skorks.com/2009/09/the-secret-of-being-a-great-mentor/, Alan Skorkin, 2009. Luettu 24.3.2011. Osa laajempaa osaamisen kehittämistä käsittelevää artikkelisarjaa.

Jos tuntuu siltä, että ajatuksena käsityöläisyys on hyvä, mutta että sen soveltaminen tämänhetkisessä työympäristössä ei ole mahdollista, ovat avoimen lähdekoodin projektit oiva mahdollisuus päästä kokeilemaan sitä. Etsi jokin itseäsi kiinnostava avoimen lähdekoodin projekti. Hyvän projektin tunnistaa siitä, että projektin ympärillä käytävä keskustelu on vilkasta ja että sen ydinporukka koostuu noin 2-3 henkilöstä, jotka saavat tulosta aikaiseksi, eli julkaisevat hyvälaatuista ja toimivaa ohjelmistoa. Liity projektiin ja seuraa projektia hetken. Aloita osallistuminen esim. tekemällä virhekorjauksia sekä osallistumalla projektin ympärillä käytävään keskusteluun. Tulet huomaamaan kohtalaisen nopeasti, että hyvän projektin ydinhenkilöistä löytyy nopeasti mestaritason ihmisiä, jotka usein ovat valmiita toimimaan mentorina. ■

Erkki Pöyhönen

Erkki toimii vanhempana konsulttina Tieto Finland Oyj:ssä ja kouluttaa aktiivisesti mm. FYIn intensiiviseminaareissa.
[erkki.poyhonen@tieto.com]

Petteri Puurunen

Petteri toimii projektijohtajana Tieto Finland Oyj:ssä ja liiketoimintaprosessien aikuiskouluttajana Haaga-Helia AMK:ssa sekä PrOSY:n vetäjänä Sytyke Ry:ssä.
[petteri.puurunen@tieto.com]

Ohjelmistojen tuottajan ammattitaito, ylpeys ja laatu

Yksilötason ammattiylpeys syynissä

Ohjelmistoalan ammattilaisuus voidaan yhdistää termin ”craftmanship” – vapaasti käännettynä ”käsityöläiseen”. Se, mikä suunnitellaan ja toteutetaan koodina on melkein kuin taidetta - tosin hyvin loogista sellaista. Käsityöläisyys saa uuden merkityksen modernin aikakauden työläisen – teknisen asiantuntemuksen – siirtämisessä asiantuntijan aivoista teknisen ongelman ratkaisuksi osaksi yrityksen kriittisen järjestelmän toimivuutta. Erään sulautettujen ohjelmistojen valmistajan puhelimessa kuvaviestin lähettämiseen tarkoitetussa puhelimessa oli yksittäinen bugi. Virhe havaittiin demovaiheessa ennen toimialan messuja. Moderni ”käsityöläinen” kutsuttiin kesälomilta ratkaisemaan aiheuttamansa ongelma ja tuotemarkkinoinnin ammattilaisten armeija pääsi onnellisena promoomaan uutta luuria.

Tarkkuus työssään on yksi tämän ammattiryhmän keskeisimmistä työn piirteistä.

Ammattiylpeyden resepti

- asenne korkeaan laatuun
- kyvykkyys loogiseen ajatteluun
- peruskoulutus
- vaatimusten ymmärtäminen
- toimialan ymmärrys
- ohjelmointikielen/kielien
- ympäristöjen tuntemus
- opitut ratkaisumallit
- yleinen kokemus

Miten sen ammattiylpeyden saa näkymään yhtä hyvin nopeassa kiinteään aikatauluun sidotussa projektissa kuin ”piikki auki” (time& material) – tyyppisessä projektissa. Väittäisin, että ammattiylpeys on asenne, joka saa tekemään parhaansa molemmissa tyypeissä. Tosin toivottavaa on kehittäjän tuottamien ratkaisujen lisäksi pragmaattisuus – tuotetaan ratkaisuja, ei jäädä leipomaan ja testaamaan koodia tästä ikuisuuteen (perfektionistin ongelma?).

Käsityöläisen yksi taitoalue on määrittely, suunnittelu, toteutus ja testausvaiheiden ammattimainen hallinta. Määrittelyvaiheessa asenne, kokemus ja muu osaaminen saavat näkyä. Menetelmästä riippumatta projektin asiakas saa kyllä nopeasti kuvan ammattitaidosta, jolla yksilö tuottaa vaatimuksia listalle, tekee ratkaisukuvauksia, integroi suusanallisesti erilaisia sanomaväyliä toisiinsa tai luo käyttötarinoita kuin pyssyn suusta.

Suunnittelussa ja toteutuksessa yksilön kyky tuottaa koodia riveittäin lienee yksi näkökulma asiaan varsinkin kun ohjelmoija työskentelee tutussa ympäristössään. Jos laatu on virheettömyyttä, niin tässä sen pitäisi näkyä. Mitä paremmassa vireessä ampujahiihtäjä on, sitä varmemmin ne taulut sieltä tipahtelevat. Toinen näkökulma on tuottaa luovia, loogisia ja pragmaattisia ratkaisuja uusilla välineillä uusille alustoille laadukkaasti. Jälkimmäinen näkökulma erottelee jyvät akanoista. Ammattitaitoa on toki tehdä ja kokeilla erilaisia ratkaisuja, kunnes päästään optimaaliseen ratkaisuun järjestelmän kannalta. Laadun mielestämme määrittää kyky löytää ratkaisu siihen, mitä ohjelman pitäisi itse asiassa tehdä käyttäjän kannalta.

Testaus tai hienolta nimeltään ”laadunvarmistus” on must ja tulee kuulua kehittäjän työhön saumattomasti. Omien virheiden poistaminen ja haku kertoo ammattimaisuudesta – tarkastella kriittisesti omaa työtään ja löytää ja poistaa sieltä virheitä, jotka eivät asianomaiseen ohjelmaan kuulu.

Esimerkit

Miten ohjelmistoammattilainen saadaan tuottamaan hyvää koodia laadukkaasti?

Parempaa koodia ja laatua ryhmässä

Agiileissa menetelmissä on varmasti hyviä puolia. Yksi parhaimmista on amerikkalainen mentaliteetti – ”minimize waste time and do the right things”. Selkeä roolitus ja vastuunjaon jalkauttaminen tiimille omista tekemisistään ovat jokaisen projektin a ja o. Sitten tulee se taas realiteettitarkistus, eli kyllä jokainen projekti vaatii tietyn joustavuusfaktorin menetelmästä huolimatta. Tulee se sitten ihmisistä, firman tai pienen putkan omasta projektikulttuurista tai firman soveltamasta menetelmästä (esim. sovellettu versio RUPista tai Scrumista). Jos talossa on vahva projektikulttuuri, ohjausmenetelmiä voidaan ripotella agiilien menetelmien ohjaukseen edistämään tai hidastamaan luovan koodin tuottamista.

Ryhmässä tekeminen riippuu ryhmän dynamiikasta, taitotasosta ja ambitiosta. Nuori vasta koulun penkiltä tempaistu asiantuntijaryhmä käyttänee käyttöliittymän vääntöön ”piin” verran enemmän aikaa kuin alkuperäinen suunnitelma. Vaikka kuinka agiilisti väännettäisiin. Laatu voi olla yhtä hyvää kuin konkareilla. Oppimiskäyrä lähtee heti nousuun ja tuottavuus roolien selvittyä ja tehtävien tullessa tutummiksi. Valmisto-ohjelmistojen tapauksissa koodaritiimin kokemus, johtavan asiantuntijan ympäristöjen tuntemus, asiakkaan kuuntelemisen taito ja tiimin sisäinen kommunikointi ovat mielestämme eniten laatuun vaikuttavia tekijöitä. Ammattiylpeys kasvaa positiivisten kokemusten kautta- yksittäin tai ryhmässä koodattujen – tai pikemminkin ratkaistujen teknisten ongelmien kautta.

Hajautetut tiimit luovat oman haasteensa yksilöille, ryhmälle ja koko projektille. Yksittäinen koodari voi ”puhua” työn jälkensä kautta ja näyttää taitotasonsa muulle ryhmälle. Mutta yhteisen kielen tulee toimia – yhä etenevässä – määrin myös suullisen kommunikaation tasolla. Ammatillista ylpeyttä tulisi olla miettiä, miten tehdä koodia laadukkaasti seuraavaan vaiheeseen ja laadun varmistuksen kautta antaa rakentavaa palautetta korjattavista bugeista. Ryhmätason koodin laatu on tietysti tekijöidensä summa ja viime kädessä vastattava siihen, mitä asiakas pyysi tekemään.

Laatu tulosyksikkötasolla

Laadun tulosyksikkötasolla määrää väistämättä kyky ohjata kokonaisuutta ja koodaamisen kulttuuri. Koko koodarien lauman ammattiosaamisen summaaminen on haastavaa, jopa jenkkifirmojen tulosorientoituneessa kulttuurissa. Summalasku kertautuu toki kulttuurissa, ovatko firmasi koodarit innostuneita uusista projekteista, asiakkuuksista, ohjelmointikielistä, ympäristöistä, hajautetuista tiimeistä tai urakierrosta? Vai vaivaako 9-17 työasenne ja moraali?

Laatua voidaan mitata projekteihin käytetyllä tuntimäärällä ja verrata toteutuneita samalla teknologia-alustalla tehtäviä projekteja toisiinsa. Toki tarkastelun perspektiivi täytyy pitää: kiinteän projektin laatu on aina heikommalla pohjalla kuin urakkatyönä tehty ilman sanktioita. Tässä täytyy muistaa koodarin ja projektin pahin vihollinen: oma myyjä, joka tahtoen tai tahtomattaan myy, mikä poliittisesti on sopivinta, tekee handoverin toimitukselle ja perävalot vinkuvat jälleen kerran. No, sitä saa, mitä tilaa –yleensä.

Tulosyksikössä seurataa toki projekteihin käytettyä aikaa, laatua, kustannuksia ja asiakastyytyväisyyttä. Järkevän projektiporftolion muodostumiseen ja ohjaamiseen vaikuttavat monet tekijät: hankkeiden ja projektien luonne, kompleksisuus, arkkitehtien ja kehittäjien kyvykkyys, resurssointitilanne, asiakkaan ambitiot ja järjestelmien luonne. Terveellistä portfoliota mitataan jatkuvasti ja tilanteisiin otetaan aktiivisesti kantaa – suomeksi ”johdetaan”.

Laatuorientoituneen yksikön tunnistaa siitä, että hanketoimiston vetäjä johtaa visiolla ja ohjaa toimintaa keskitetysti, hallitsee omat sidosryhmänsä sivulle (arkkitehtitoimisto, infratoimisto) ja ylöspäin (asiakkaat, omat liiketoiminnan lähettiläät). Hanketoimisto verkottaa tehokkaasti työnjohdon ja kehittäjät, käyttää viimeisimpiä somen työkaluja viestintään, luo urapolut, mahdollistaa joustavan työssäoppimisen ja formaalin koulutuksen. Jos yrityksesi ei tunne ”train the trainer” – konseptia – niin tervetuloa 2000-luvulle. Ja takaisin laatuun – terveellinen laatujohtaminen ja käytännöt näkyvät auditoinneissa ja asenteissa. ▪

Checklist
laadun tuottaminen yksilötasolla

- ✓ ammattiylpeys
- ✓ kokemus, tietotaito
- ✓ kyvykkyys
- ✓ tietämyksen siirron kypsyys, tiedon jako, projektikulttuuri
- ✓ tiimityötaidot

Checklist
laadun tuottamiselle ryhmässä

- ✓ kick-off
- ✓ perehdytys
- ✓ pelisäännöt
- ✓ tiedon jako, tietämyksen ja hiljaisen tiedon siirto
- ✓ menetelmä
- ✓ yhteinen kieli
- ✓ raportoinnit
- ✓ tapaamiset
- ✓ jokainen ”myy” – asenne, jenkkifirmat – ammattiylpeyttä myydä omalla työllään, isoissa firmoissa mangutaan ”projektikoodin”perään – pienissä koodin leipomofirmoissa – koodaajat ovat moniosaajia…

KUUTAMOLLA

Terveisiä Softasuurkiltamestarilta!

Kuvitelkaa mielessänne sateisen harmaa myöhäissyksyn ilta. Vanhan kaupungin kosteiden kivimuurien vierellä, holvikaaristen ikkunoiden rivistön alla liikkuu vaitonainen harmaisiin kaapuihin pukeutuneiden huppupäiden jono. Hahmoilla on munkeilta tutut kaavut, joissa vyön virkaa toimittaa kaavunharmaa punottu naru ja hahmot kulkevat lakonisen eleettömästi, kädet piilotettuina laajojen hihojen muodostamaan muhviin.

Ääneti jono etenee sateisilla kaduilla, kunnes äkkiarvaamatta ensimmäinen hahmo kääntyy alaspäin viettäviin portaisiin ja jyskyttää nyrkillään portaiden juuressa olevaa jykevää raudoitettua tammiovea, muun jonon pysähtyessä paikalleen. Ovesta aukeaa pieni tähystysluukku ja hetken sen jälkeen luukku sulkeutuu ja ovi avautuu. Hahmojono häviää synkeästä oviaukosta sisään ja ovi sulkeutuu jyhkeästi kumahtaen. Vain tihkusade jää yksin kadulle.

Sisällä tiiliseinäisessä aulassa hahmojoukko kokoontuu yhteen oven avanneen miehen ympärille. Mies odottaa, että viimeinenkin huppupäinen hahmo on päässyt lähietäisyydelle. Sen jälkeen hän sanoo hieman jännittyneellä äänellä: ”Tervetuloa Sytyke ry:n koodarikillan kokoukseen!”

Tällaiseksiko se on maailma menossa? Siltä se vähän vaikuttaa, jos katsoo Software Craftmanship –liikkeen kasvua. Leviämisen suhteen liike alkaa muistuttaa jo melkein uskonlahkoa tai sikainfluenssaa. Joku kyynisen maailmankatsomuksen omaava kolumnisti voisi happaman ihailevasti sanoa, että kappas, tuollaista hypetystä ei vielä kukaan ole keksinytkään. Että vartavasten pyritään näpertämään ja tekemään hitaasti ja tehdään se vielä ylpeillen siitä! Mutta ehkäpä kyse ei olekaan hitaasti tekemisestä. Laatua ei synny hutiloiden ja aika on rahaa? Joten, jos laadun tekemiseen menee aikaa, niin kallista siitä tulee. Monella ihmisellä on kummallinen hinku saada laadukasta tavaraa, joten nyt sitten herää kysymys, että miksi liiketoiminnassa lähes aina valitaan se halvin? No tietenkin siksi, että yrityksen ei tarvitse keikaroida. Yrityksen tehtävä on tuottaa rahaa ja sitä tehdessään minimoida rahanansaitsemiseen tarvittavia kuluja. Kuten palkkakulut. Joita taas oletettavasti ohjelmistokäsityöläiset haluaisivat nostaa profiloidessaan oman tuotantopanoksensa osaamista korostaen.

Tässä on ristiriita. Mutta siihen on ratkaisu, jonka pyyteettömästi tarjoan kaikille ohjelmistoalan ja systeemityön ammattilaisille. Ratkaisu löytyy historiasta. Keskiajalla kiltalaitos pyrki pitämään tarkkaa kontrollia ja hankkiutumaan monopoliasemaan eri ammattikuntien suhteen. Palauttakaamme tuo jalo aate! Ensiksi julistan itseni täten Suomen softasuurkiltamestariksi! Kaikki maamme Java-koodarit, tietokantaadminit, käyttöliittymäsuunnittelijat, testaajat, projektipäälliköt ja muut sellaiset, järjestäytykää omaan kiltaanne ja pitäkää huoli, että ammattianne ei harjoita kukaan kiltanne ulkopuolelta. Sitten valitsette kiltamestarin, joka ilmoittautuu minulle. Kiltaanhan pääsee mukaan vain ammattilaiset eli mestarit. Oppipojat pääsevät mukaan sitten, kun tekevät mestarinäytteen, jota voi yrittää sitten, kun riittävän kauan on ollut mestarin opissa. Mikäli kiltanne ei oikein pysty hoitamaan asioitaan, eli jos epäilette, että killallenne kuuluvia töitä tekee joku ulkopuolinen, niin voin perustuttaa Ympäripuhuja-killan, jonka jäsenet mielellään suostuttelevat joko työnantajan käyttämään ammattilaista tai ei-ammattilaisen lopettamaan. Ympäripuhujat voisivat käyttää vaikka nahkaliivejä, joissa olisi killan tunnus selässä. Jotta järjestys olisi taattu ja hallinnolliset kulut tulisivat peitetyksi, niin kiltalaiset maksaisivat nimellistä, kymmenen prosentin maksua palkkatuloistaan Softasuurkiltamestarille, joka sitten käyttäisi rahat kiltajärjestelmän kehittämiseen. Olisin pitkämielinen ja oikeudenmukainen Jum.. suurkiltamestari ja kiltalaitoksen alaisuudessa olijat voisivat elää rauhan ja turvan ympäröimänä, kehittäen osaamistaan entistä korkeammalle.

Mutta aivan ensimmäisenä kaikki kiltalaitoksen uhkaajat ja kilpailijat on tuhottava. Eli Sytyke on lopetettava! ▪

Sytyke Ry:n Hallitus 2011

Hallitukseen kuuluu puheenjohtajan lisäksi 6 varsinaista jäsentä ja 2 varajäsentä.

Sytyke ry yhdistää suomalaiset ohjelmistokehityksen asiantuntijat. Yhdistyksen muodostavat erikoisalueisiin keskittyvät osaamisyhteisöt, joiden jäsenet välittävät toisilleen omien alojensa syvällisintä tietoa ja tuoreimpia trendejä.
Sytyke ry:n toiminta tapahtuu pääosin osaamisyhteisöissä, joiden tavoitteena on auttaa jäseniään kehittymään yhä vahvemmiksi omien alojensa ammattilaisiksi.
Sytyke ry on Tietotekniikan liiton (TTL ry) suurin valtakunnallinen teemayhdistys.

Hallitus 2011

Juha Jääskinen, puheenjohtaja
Tuotepäällikkö Digia Oyj
Hiomotie 19, 00380 Helsinki
Puh. 050 4200 822
juha.jaaskinen@digia.com

Rami Talme, talousvastaava
Toimitusjohtaja RP5 Software Oy
Puh. 040 830 3919
rami.talme@rp5.fi

Mikko Holmberg, varapuheenjohtaja
CodeBakers Oy
Metsänneidonkuja 6, 02130 ESPOO
Puh. 040 741 3991
mikko.holmberg@codebakers.fi

Tarmo Toikkanen, viestintävastaava
NP-Ratkaisut Oy, Aalto-yliopisto,
T:mi Tarmo Toikkanen
PL31000, 00076 AALTO
tarmo@iki.fi

Minna Oksanen, OSY-vastaava
Senior Consultant Avarea Oy
minna.oksanen@gmail.com

Anne Isotalo, hallituksen sihteeri
SOK
PL 1, 00088 S-RYHMÄ
Puh. 050 320 3899
anne.isotalo@gmail.com

Sytyke ry:n hallituksen sähköpostilista:
hallitus@sytyke.ttlry.fi

Juho Kauppi, seminaarivastaava
Somea Oy
Puh. 040 846 4610
juho.kauppi@iki.fi

Varajäsenet

Altti Lagstedt, varajäsen
HAAGA-HELIA ammattikorkeakoulu
Ratapihantie 13, 00520 Helsinki
Puh. 040 488 7272
altti.lagstedt@haaga-helia.fi

Marianne Malila
Senior Consultant JET-kassa ry
malila68@gmail.com

Liittokokousedustajat 2011

Minna Oksanen
minna.oksanen@gmail.com

Mikko Holmberg
mikko.holmberg@codebakers.fi

Lauri Laitinen
lauri.laitinen@nokia.com

Seuraava lehti ilmestyy 5.10.2011!

CodeBakersin menetelmäkehityspalvelu

Miten toteutat hankkeitasi nopeammin, laadukkaammin ja kustannustehokkaammin?

Haluatko kehittää omaa toimintaasi tehokkaampaan suuntaan?

Liiketoiminnan vaatimusten pitää näkyä kehityshankkeessa ja laadunvarmistuksessa läpinäkyvästi. ***Vaatimusten ja määrittelyjen muutosten*** *pitää tulla samaan aikaan samanlaisina kaikille hankkeen eri osapuolille. Menetelmäkehityspalvelumme on vastaus näihin haasteisiin.*

Palvelun lopputuloksena asiakkaamme saavat organisaationsa tarpeisiin mukautetun ohjelmistokehitysprosessin, joka huomioi ***vaatimustenhallinnan ja testauksen*** *osana kokonaisuutta. Prosessi on helppo oppia ja ymmärtää ja se tarjoaa konkreettisia ohjeita, toimintamalleja ja dokumenttipohjia.*

Palvelumme perustuu ***avoimeen toimintatapaan,*** *jolla pystymme varmistamaan onnistuneen, asiakkaan odotukset ja tarpeet täyttävän lopputuloksen.*

www.codebakers.fi/menetelmakehitys

Me haluamme sinut!

Haemme menetelmäkehitystiimiimme vahvistusta. Oletko se sinä?

- **Oletko sitä mieltä, että asioita voisi tehdä fiksummin?**
- **Kiinnostaako sinua menetelmien ja käytäntöjen kehittäminen?**
- **Oletko kokenut ohjelmistotuotannon ammattilainen, joka on uusia haasteita vailla?**

Käy tutustumassa avoimiin paikkoihimme osoitteessa

www.codebakers.fi/rekrytointi

Lähetä hakemuksesi viimeistään 30.5.2011

Lisätietoja:
Seppo Takanen
050-5810140
seppo.takanen@codebakers.fi

Codebakers lyhyesti:

- **96% asiakkaistamme suosittelee meitä**
- **Asiakaskuntaamme kuuluu joukko tunnetuimpia suomalaisia yrityksiä**
- **Olemme Suomen nopeimmin kasvavien ohjelmistoyritysten joukossa**
- **Liikevaihtomme 2010 3,7 M€, 44 paakaria**
- **Johdon ja työntekijöiden omistama**
- **Sijaitsemme Espoon Tapiolassa**

CodeBakers