Samppa Hynninen

Behavior-driven development mobiiliohjelmistojen kehityksen tukena

Tietotekniikan (Ohjelmisto- ja
tietoliikennetekniikka)
pro gradu -tutkielma
24. maaliskuuta 2014

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Samppa Hynninen
Yhteystiedot: samppa.hynninen@jyu.fi
Työn nimi: Behavior-driven development mobiiliohjelmistojen kehityksen tukena
Title in English: Using behavior-driven development to support software development on mobile platforms
Työ: Tietotekniikan (Ohjelmisto- ja tietoliikennetekniikka) pro gradu -tutkielma
Sivumäärä: 19
Tiivistelmä: Tähän tulee tutkielman tiivistelmä.
Abstract: Here comes the abstract of the thesis.
Avainsanat: bdd, behariour-driven development, mobiilialustat
Keywords: bdd, behariour-driven development, mobile platforms

Sisältö

1 Johdanto

Lähivuosina erilaisten mobiilipäätteiden osuus kuluttajien käyttämistä laitteista on kasvanut räjähdysmäisesti. Perinteisten pöytätietokoneiden, kannettavien tietokoneiden ja matkapuhelinten väliin on syntynyt täysin uusia laiteryhmiä, kuten esimerkiksi tablet-tietokoneet. Ohjelmistoteollisuus on joutunut muuntautumaan uusiin vaatimuksiin, jotka uudenlaiset sovellusalustat ovat tuoneet mukanaan. Palvelujen tulee nykyään monesti olla käytettävissä useilla eri alustoilla ja erilaisia alustoja voivat koskea erilaiset vaatimukset. Ohjelmistokehitysmenetelmiä on jouduttu kehittämään jatkuvasti muuttuvan ympäristön ja uusien tarpeiden myötä. Behaviour-driven development, eli käyttäytymislähtöinen kehitys on yksi viime vuosina kehittyneistä ohjelmistokehitystä tukevista tekniikoista, jossa kehityksen lähtökohtana ovat määrittelyt siitä, kuinka ohjelmiston tulisi toimia ja käyttäytyä.

Ohjelmistotestauksen ollessa kriittinen osa ohjelmistotuotantoprosessia ja erityisesti laadunvarmistusta, sen merkitys korostuu entisestään, kun ohjelmistoja toimitetaan useille eri alustoille. Tässä tutkielmassa käsitellään behaviour-driven developmentia ja sen taustoja niin menetelmällisestä kuin puhtaasti teknologisesta näkökulmasta. Aluksi selvitetään mitä BDD:llä tarkoitetaan ja mistä se on kehittynyt. Taustojen ja historian ohella katselmoidaan ohjelmistokehityksen haasteita ja ongelmakohtia, joita BDD:n avulla pyritään välttämään. Teknologioiden osalta käsitellään erityisesti BDD-testaamista mobiiliohjelmistojen kehityksessä ja sen poikkeavuuksia sekä erityispiirteitä verrattuna BDD-menetelmiin muissa ympäristöissä. Tässä yhteydessä tarkastellaan erikseen eri mobiilialustojen natiivisovelluksia sekä uusiin HTML5-teknologioihin pohjautuvia alustariippumattomia sovelluksia, sillä niiden testaaminen eroaa merkittävästi toisistaan.

Tämän pro gradu -tutkielman tavoitteena on kartoittaa behaviour-driven developmentin tarjoamia mahdollisuuksia helpottaa mobiiliohjelmistojen kehitystä ja erityisesti testausta. Tarkoituksena on selvittää millaisia BDD-testikehyksiä merkittävimmille mobiilialustoille on tällä hetkellä olemassa ja miten eri mobiilialustojen testikehykset poikkeavat toisistaan. Tämä lisäksi tarkoituksena on selvittää, millaisia liiketoiminnallisia etuja BDD:llä voidaan mobiilisovelluskehityksessä saavuttaa. Tutkielma toteutetaan käyttäen konstruktiivista tutkimusotetta ja siinä pyritään lopputuloksena toteuttamaan ympäristö, jossa samaa sovellusta voitaisiin testataan eri mobiilialustoilla yhdellä testikokoelmalla. Testiympäristön toteutuksen yhteydessä

tutkitaan millaisia puutteita nykyratkaisuissa on ja miten niitä voitaisiin kehittää, jotta yhden testikokoelman mallia voitaisiin käyttää.

2 Ketterä ohjelmistokehitys

2.1 Taustaa ja ketterien menetelmien kehitys

Niin kauan kuin tietokoneohjelmistoja on kehitetty, on niiden yhteydessä kehitetty myös erilaisia malleja, joiden tarkoituksena on helpottaa ja sujuvoittaa ohjelmistokehitystä. Menetelmät voidaan karkeasti jakaa inkrementaalisiin ja iteratiivisiin menetelmiin [1]. Inkrementaalisissa malleissa ideana on se, että ohjelmisto kasvaa jatkuvasti kehittyen kohti lopullista valmista tuotetta, kun taas iteratiivisissa malleissa ohjelmistoa kehitetään pienissä osissa lyhyemillä aikaväleillä ja tätä työtä toistetaan useaan otteeseen. Seuraavassa esitelläänkin muutamia tunnettuja ohjelmistotuotannon malleja ja niiden kehitystä kohti ketteriä menetelmiä.

2.1.1 Vesiputousmalli

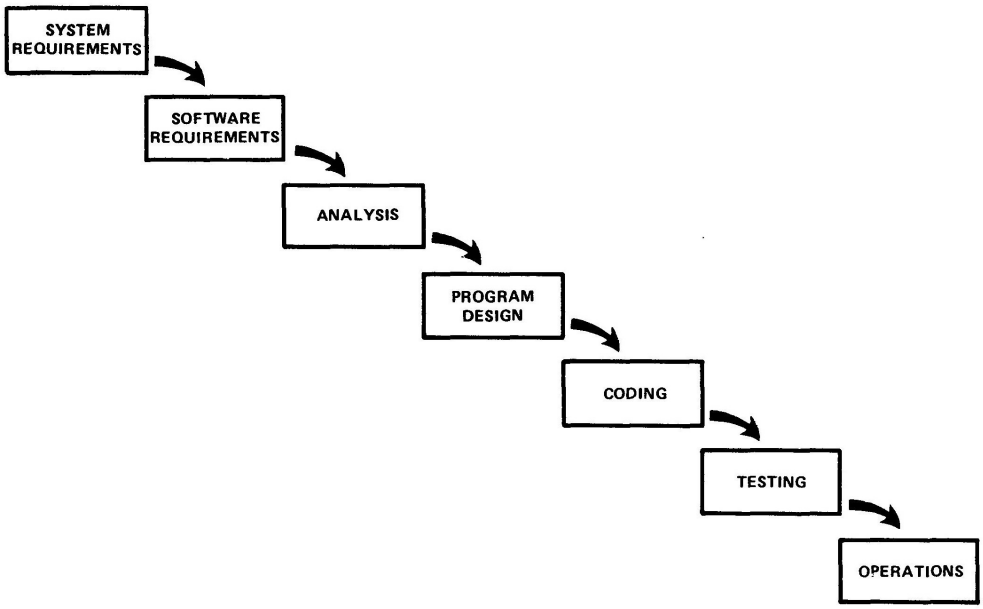
Yksi varhaisista ohjelmistokehityksen malleista on inkrementaalinen vesiputousmalli, jonka perustana toimii hyvin pitkälti Winston Roycen kuvaama malli artikkelissaan Managing the development of large software systems [3] vuodelta 1970. Huolimatta siitä, että vesiputousmalli on erittäin vanha ja sen käyttäminen sisältää tunnettuja riskejä, on malli edelleen hyvin suosittu. Roycen malli perustuu ajatukselle, että kaikissa ohjelmistokehitystehtävissä on aina ainakin kaksi vaihetta, analyysivaihe ja itse ohjelmiston koodausvaihe. Vesiputousmallissa luodaan tämän ajatuksen ympärille laajempi malli, jossa edellämainittuja vaiheita on täydennetty uusilla vaiheilla, jotka ovat kuitenkin täysin erillisiä analyysi- ja koodausvaiheista. Näitä ovat Roycen mallissa [3] kaksi vaatimusmäärittelyvaihetta, ohjelmiston suunnittelu ja koodauksen jälkeinen testausvaihe. Näillä lisäyksillä malli on soveltuvampi suurten ohjelmistoprojektien läpivientiin. Roycen esittämä malli ei kuitenkaan ole täysin se vesiputousmalli, joka on vielä nykyäänkin käytössä, sillä Royce itsekin ehdottaa parhaimpana versiona mallistaan sellaista, jossa eri vaiheiden välillä voidaan liikkua kumpaankin suuntaan, eli myös takaisinpäin edellseen vaiheeseen.

Royce kuitenkin tiedostaa vesiputousmallin sisältämiä riskejä jo omassa artikkelissaan. Yksi näistä on esimerkiksi vaara, että vasta testivaiheessa huomataan puutteita, jotka johtavat perinpohjaisiin muutoksiin ohjelmiston rakenteessa tai toiminnassa, jolloin käytännössä koko kehitysprosessi joudutaan aloittamaan alusta.

Näitä puutteita voi suurellakin todennäköisyydellä ilmetä, sillä testausvaihe on ensimmäinen kerta, kun järjestelmä toimii kokonaisuutena integroituna muihin järjestelmiin [3]. Toinen merkittävä tekijä, joka monesti jätetään huomioimatta vesiputousmallia käytettäessä, on se, että Royce itsekin suosittelee käymään prosessin läpi kahteen kertaan. Tämä pätee etenkin tilanteisiin, jossa toimitetaan asiakkaalle jokin täysin uusi tuote. Tällöin ensimmäisellä kerralla luodaan pilotti tuotteesta lyhyessä ajassa. Tämän pilotin tarkoituksena on tuoda esiin kehitettävän tuotteen erityiset haasteet, löytää suunnittelun mahdolliset puutteet sekä luoda eri vaihtoehtoja näiden ratkaisemiseen [3]. Tällöin lyhyessä ajassa kehitetty pilottituote toimii ikään kuin koko projektin simulaationa, jolloin vaikeisiin tilanteisiin osataan varautua paremmin.

Royce ei itsekään pitänyt malliaan parhaana mahdollisena suurten ohjelmistojen kehitykseen, vaan Larmanin ja Basilin artikkelissa [1] kerrotaankin, kuinka hän itsekin toteaa vesiputousmallin olevan kaikista yksinkertaisin malli, joka ei kuitenkaan todennäköisesti tule toimimaan kuin kaikkein yksinkertaisimpien ja suoraviivaisimpien projektien yhteydessä. Ensimmäiset maininnat ja kehotukset iteratiiviseen ohjelmistokehitykseen ovatkin jo ajalta ennen Roycen mallin julkaisua. Jo vuonna 1969 IBM:n tutkimuskeskuksessa M.M Lehman kuvasi iteratiivista mallia, jossa erotettiin suunnittelu, arviointi ja dokumentointi [1]. Tässä mallissa suunnittelu tuotti jo toimivan mallin, jota sitten iteratiivisesti kehitettiin eteenpäin.

Kuitenkin vielä nykypäivänäkin suuriakin ohjelmistoprojekteja toteutetaan käyttäen vesiputousmallia, sillä sen koetaan sisältävän joitain etuja tietyissä ohjelmistoprojektin vaiheissa verrattuna ketteriin menetelmiin. Monesti nämä ratkaisevat edut liittyvät erityisesti projektien myyntiin tarjousvaiheessa tehtyjen tarjousten muotoiluun. Vesiputousmallilla myytävissä ratkaisuissa on helpompaa myydä asiakkaalle kiinteällä hinnalla yksi tarkasti määritelty kokonaisuus, jonka myötä asiakas kokee tietävänsä tarkalleen, millaisen tuotteen hän on saamassa ja mihin hintaan. Ketteriä projekteja myytäessä asiakas käytännössä ostaa vain työtä, eikä kiinteästi määritettyä lopputulosta [2]. Tällöin asiakkaalle jää jatkuvasti mahdollisuus vaikuttaa siihen, että tehty työ on juuri sitä, mistä hänelle on eniten hyötyä, eikä ohjelmistoon toteuteta esimerkiksi täysin turhia tai vanhentuneita vaatimuksia. Kuitenkin tästä huolimatta monesti ainoastaan lopullinen kiinteä hinta ja myyntivaiheessa määritelty lopputulos ratkaisevat ostopäätöstä tehdessä, jolloin edelleen nykyään saatetaan päätyä toteuttamaan projekteja vesiputousmallia käyttäen.



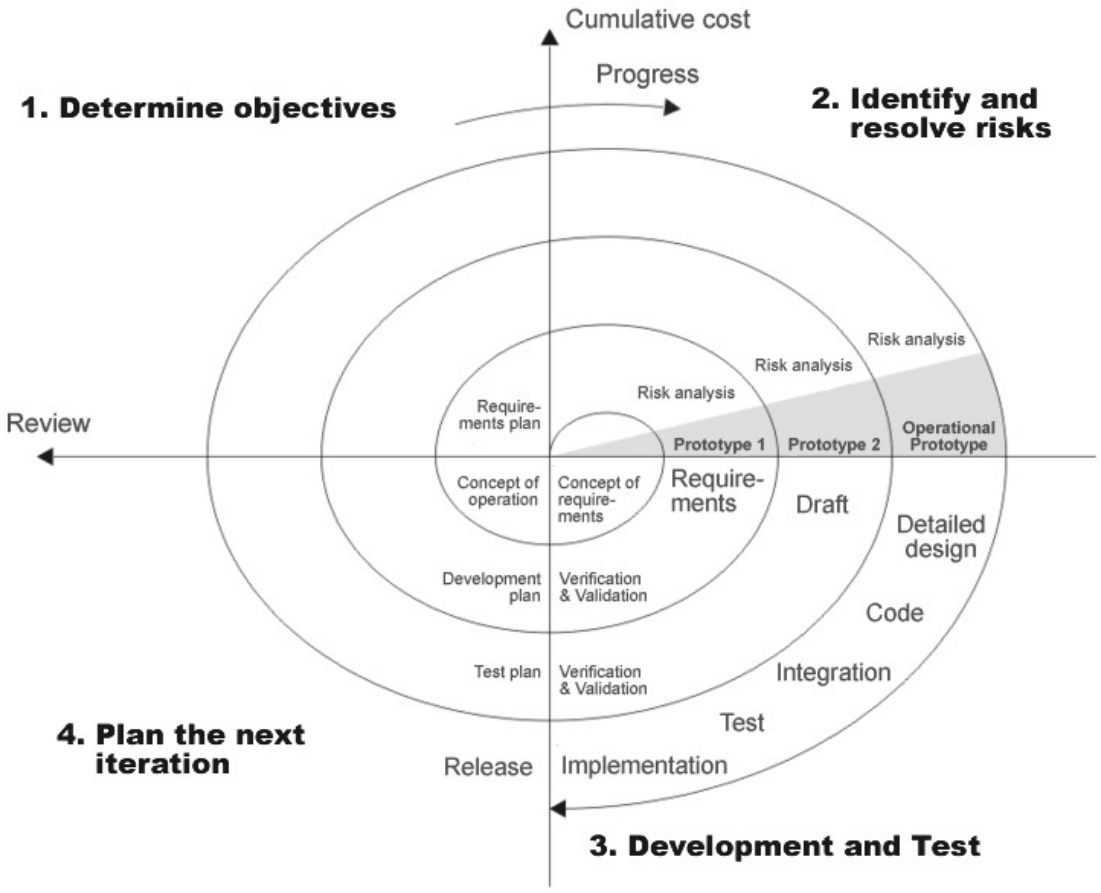
Kuva 2.1: Vesiputousmalli [3]

2.1.2 Spiraalimalli

Spiraalimalli on Barry Boehmin kehittämä [4] malli vuodelta 1986, joka on ensimmäisiä tunnettuja iteratiivisia malleja ohjelmistokehityksessä. Spiraalimallin tärkeimpänä ajatuksena on se, että siinä otetaan riskit entistä paremmin huomioon verrattuna esimerkiksi perinteiseen vesiputousmalliin. Spiraalimallissa jokainen iteraatio sisältää neljä merkittävää osaa, jotka ovat [4]:

- Iteraation tavoitteiden, vaihtoehtojen ja rajoitteiden määrittely
- Eri vaihtoehtojen arviointi sekä riskien tunnistaminen ja arviointi
- Kehitystyö ja seuraavan tason tuotteen määrittely
- Seuraavan iteraation suunnittelu

Mallissa jokaisessa iteraatiossa toteutetaan riskianalyysin jälkeen prototyyppimalli, josta sitten lähdetään kehittämään valmista tuotetta, aivan kuten Roycen [3] ehdottamassa pilottituotemallissa. Lisäksi spiraalimallissa merkittävää on se, että jokainen iteraatiokierros sillä hetkellä valmiina olevan tuotteen arviointiin, johon osallistuvat kaikki tärkeimmät henkilöt, jotka tulevat käyttämään tuotetta [4]. Boehmin malli toimiikin pohjana hyvin monille nykyisille ketterille menetelmille, joissa hyödynnetään samankaltaisia iteraatioita. Esimerkiksi seuraavana käsiteltävässä



Kuva 2.2: Spiraalimalli [3]

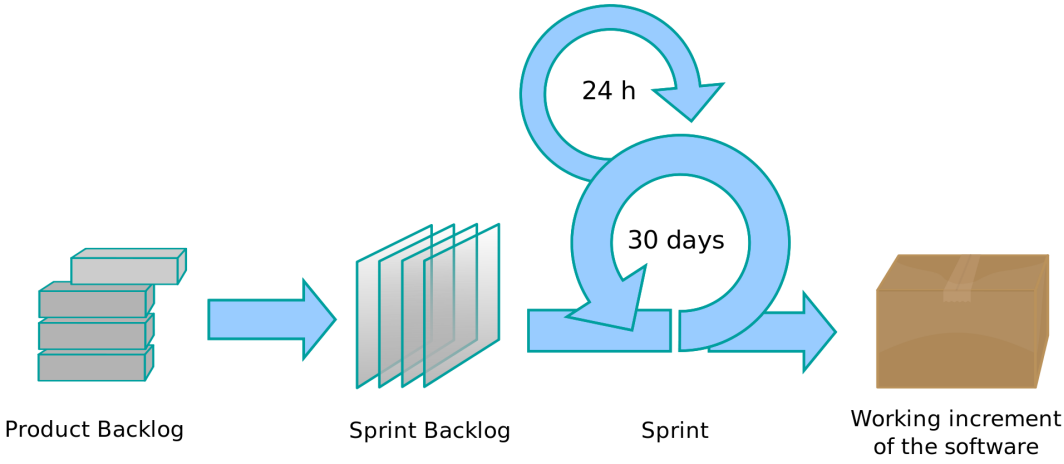
Scrumissa sprintit vastaavat hyvin pitkälle spiraalimallin iteraatioita, eli Scrum käytännössä sisältää spiraalimallin prosessit, joiden päälle on vielä vain lisätty muuta.

Spiraalimallissa käytetään erittäin paljon riskiä määrittämään tehtäviä asioita. Boehmin mukaan [4] esimerkiksi eri työvaiheisiin käytettävä aika määräytyy sen perusteella, miten ne vaikuttavat olemassaolevaan riskiin. Jos esimerkiksi testaukseen käytetty aika pienentää projektin kokonaisriskiä, siihen kannattaa käyttää aikaa mieluummin kuin esimerkiksi uuden sovelluskehyksen käyttöön vanhan tutun sovelluskehyksen sijaan, sillä tällainen valinta vain lisäisi projektin riskiä ja epäonnistumisen mahdollisuutta. Tämän lisäksi Boehmin mukaan projektiryhmä käyttää riskin arviointia siihen, kuinka yksityiskohtaisella tasolla asioita kannattaa projektissa tehdä [4]. Esimerkiksi projektin dokumentointi voisi olla yksi osa-alue, johon tätä voidaan soveltaa. Projektiryhmän tulee tällöin tehdä päätös, milloin projektin dokumentaatio on riittävällä tasolla siten, että sen tarkentamisesta ja edelleen kehittämisestä ei saada enää mitään hyötyä, vaan se voi johtaa vain kasvavaan riskiin, kuten esimerkiksi julkaisun myöhästymiseen.

Uudemmassa julkaisussaan vuodelta 2000 Boehm [5] vielä erikseen listaa merkkejä, jotka erottavat projektin spiraalimallista ja ovat ennenkaikkea haitallisia projektin onnistumisen edellytyksille. Käytännössä nämä merkit ovat sellaisia, joista voidaan huomata käytettävän oikeasti vesiputousmallia, jota vain nimitetään spiraalimalliksi. Tällaisia merkkejä ovat Boehmin mukaan seuraavat kuusi olettamusta [5]:

- Vaatimukset tunnetaan ennen ohjelmiston toteutusta
- Vaatimuksissa ei ole epäselviä olettamuksia
- Vaatimusten luonne ei muutu kehitystyön ja projektin etenemisen myötä
- Vaatimukset täyttävät kaikkien sidosryhmien edustajien oletukset
- Oikea arkkitehtuuri vaatimusten täyttämiseen on hyvin ymmärretty ja omaksuttu
- Tarpeeksi kalenteriaikaa on varattu, jotta voidaan edetä peräkkäisessä järjestyksessä vaihe kerrallaan

Boehm määrittelee myös erilaisia invariantteja, jotka tulee olla olemassa spiraalimallin toimimisen edellytyksenä [5]. Nämä invariantit käytännössä määrittelevät vain tarkemmin spiraalimallin eri osa-alueet ja niiden sisällöt. Näihin Boehmin invariantteihin kuuluu esimerkiksi edellämainitut käytettävän panoksen määrittely



Kuva 2.3: Scrum [7]

riskin perusteella sekä työvaiheen toteutuksen yksityiskohtaisuuden tason arviointi riskin perusteella. Näiden lisäksi Boehmin määrittelemiin invariantteihin lasketaan myös keskittyminen järjestelmään ja sen elinkaareen sen sijaan, että ainoastaan mietittäisiin ohjelmiston kehittämistä teknisestä näkökulmasta [5].

2.1.3 Scrum

Scrum on moderni iteratiivinen ohjelmistokehityksen viitekehys, jota ovat kehittäneet pääasiallisesti yhdysvaltalaiset Ken Schwaber ja Jeff Sutherland aina 1990-luvun alkupuolelta lähtien. Kuten termi viitekehys antaa ymmärtää, ei Scrum tarjoa mitään yksityiskohtaista prosessia ohjelmistoprojektin toteuttamiseen, vaan ennemminkin juuri kehyksen, jonka puitteissa voidaan hallita monimutkaisten tuotteiden kehitystä [6].

2.2 TDD

TDD, eli Test-driven development, tarkoittaa testivetoista ohjelmistokehitystä. TDD poikkeaa edellä esitetyistä malleista siinä, että siinä missä edellä esitellyt mallit kattavat koko ohjelmistokehitysprosessin, ottaa TDD kantaa ainoastaan edeltävissä malleissa esiintyneisiin käytännön toteutus- ja testausvaiheisiin. Käytännössä siis nykyään ohjelmistoprojekteissa valitaan jokin koko ohjelmistokehitysprosessin kattava malli, kuten Scrum, jossa sitten hyödynnetään TDD:tä ja sen ohjelmointikäy-

tänteitä. Seuraavassa esitelläänkin TDD:n keskeisimmät ideat ja toimintaperiaatteet, joita käytännön ohjelmistokehityksessä usein noudatetaan.

2.3 ATDD

3 Behaviour-driven development

3.1 BDD:n lähtökohdat ohjelmistokehitystä tukevana menetelmänä

3.2 Teknologioista

4 BDD:n tarjoamat liiketoiminnalliset edut

4.1 Asiakkaan käyttäjätarinoista hyväksymistesteiksi

4.2 BDD:n hyödyntäminen mobiilikehityksessä offshore-tiimeillä

5 BDD mobiilialustoilla

5.1 Natiivisovellukset

5.2 HTML5-sovellukset

6 Crossplatform-testaaminen eri mobiilialustoilla

6.1 iOS BDD-frameworkit

6.2 Android BDD-frameworkit

6.3 Windows phone BDD-frameworkit

6.4 Mahdollisuudet testata kaikki alustat yhdellä testisetillä

7 Yhteenveto

Yhteenveto tähän

Lähteet

[1] Craig Larman ja Victor Basili, *Iterative and incremental developments. a brief history*, IEEE Computer Society, Volume: 36, Issue: 6, 2003.

[2] Valtiokonttori, Valtion IT-palvelukeskus, *Ketterän palvelukehityksen ostaminen*, 2013.

[3] Winston Royce, *Managing the development of large software systems*, Proceedings of IEEE WESCON, 1970.

[4] Barry Boehm, *A spiral model of software development and enhancement*, ACM SIGSOFT Software engineering notes vol 11 no 4, 1986.

[5] Barry Boehm, *Spiral Development: Experience, Principles and Refinements*, Spiral Development Workshop, February 9, 2000.

[6] Ken Schwaber ja Jeff Sutherland, *The Scrum Guide*, Scrum.org, 2013.

[7] *Scrum process*, saatavilla WWW-muodossa <URL: `http://en.wikipedia.org/wiki/File:Scrum`$_p$*rocess.svg* > , *viitattu*24.03.2014.

[8] *Agile Manifesto*, 2001, saatavilla WWW-muodossa <URL: `http://agilemanifesto.org/`>, viitattu 12.02.2013.

[9] Matt Wayne ja Aslak Hellesøy, *The Cucumber Book, Behaviour-Driven Development for Testers and Developers*, Pragmatic Programmers LLC, 2012.

[10] Carlos Solís ja Xiaofeng Wang, *A Study of the Characteristics of Behaviour Driven Development*, 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 2011.

[11] Dan North, *Introducing BDD*, 2006, saatavilla WWW-muodossa <URL: `http://dannorth.net/introducing-bdd/`>, viitattu 27.01.2013.

[12] Gojko Adzic, *Specification by Example*, Manning Publications Co, 2011.

[13] Gojko Adzic, *Bridgind the Communication Gap, Specification by Example and Agile Acceptance Testing*, Neuri Limited, 2009.

[14] Susan Hammond ja David Umphress, *Test driven development: the state of the practice*, ACM-SE '12 Proceedings of the 50th Annual Southeast Regional Conference, s. 158-163, 2012.

[15] Chris Rimmer, *Introduction, Behaviour-Driven Development*, 2010, saatavilla WWW-muodossa <URL: `http://behaviour-driven.org/Introduction`>, viitattu 27.01.2013

[16] IBM developerWorks: http://www.ibm.com/developerworks/java/library/j-cq09187/index.html

[17] RSpec, Cucumber, JBehave, Robotium, Selenium jne.