Test Driven Developement-menetelmän tehokkuus ja laatu

Petri Pihlajaniemi

Referaatti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 30. tammikuuta 2015

Test Driven Development

Test Driven Development (TDD) on Kent Beckin esittelemä ohjelmistokehityksen suunnittelumalli, jossa kehittäjä kirjoittaa automaattiset testit ennen koodia. Testin kirjoittamisen jälkeen kehittäjä kirjoittaa ohjelmakoodin jota pidetään hyväksyttävänä vasta, kun se läpäisee uuden ja kaikki vanhat testit. TDD:tä käyttäessä ei vaadita tarkkaa suunnittelua, vaan ohjelman rakenne kehittyy iteratiivisesti ohjelman vaatimusmäärittelyn mukaan rakennettujen testien perusteella.

Perehdyin kahteen TDD:n toimivuutta käsittelevään artikkeliin, Guptan ja Jaloten *An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development*[1] sekä Wilkersonin, Nunamakerin ja Mercerin *Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development*[2]. Tutkimuksissa on arvioitu käytännön ohjelmistokehityksen perusteella menetelmän toimivuutta.

Testimenetelmät

Molemmissa kokeissa koeryhmänä on käytetty yliopisto-opiskelijoita. Guptan ja Jaloten ryhmässä oli IIT Kanpurin Advanced Object-Oriented Analysis and Modeling-kurssin 25 oppilasta, joista 22 tietojenkäsittelytieteen opiskelijoita. Kolme elektroniikan opiskelijaa jaettiin satunnaisesti tietojenkäsittelytietelijöistä muodostettuihin kahteen 11 hengen ryhmään. Opiskelijoilla oli valmiiksi vähintään kahden ohjelmointikurssin taidot ja he tunsivat tutkimuksessa käytetyn Java-ohjelmointikielen. Ryhmien tehtävänä oli kirjoittaa kaksi ohjelmaa: opiskelijoiden rekisteröintijärjestelmä (P1) ja yksinkertainen raha-automaattiohjelma (P2).

Kahden viikon aikana ryhmät käyttivät perinteistä CCD-ohjelmistokehitystä jossa he ensin suunnittelivat sekvenssi- ja luokkakaavioiden avulla ohjelman rakennetta sekä TDD-ohjelmistokehitystä. Ryhmä 1 teki ensin ohjelman P1 käyttäen CCD-menetelmiä ja ohjelman P2 TDD-menetelmällä, ryhmä 2 toimi päinvastoin. Kolmannella viikolla ryhmille annettiin valmiit testipaketit, ilmenneet virheet ohjelmakoodissa korjattiin ja raportoitiin. Oppilaat palauttivat tiedonkeräyslomakkeet ohjelmien kehitysvaiheiden ja testausvaiheen jälkeen. Lisäksi lopussa oppilaat palauttivat myös kyselylomakkeen.

Wilkersonin, Nunamakerin ja Mercerin tutkimuksessa oli Yhdysvaltalaisen yliopiston olio-ohjelmointiin ja suunnitteluun keskittyneen kurssin osaanottajat, joista kaikki ylioppilaita. 58 vapaaehtoisesta valittiin lähtotasotestin perusteella 40 korkeimman pistemäärän saanutta oppilasta koeryhmään. Tutkimukseen liittyneet tehtävät kuuluivat myös kurssin tehtäviin ja osallistujien kesken arvottiin 100 dollarin rahapalkinto, joten tutkijat uskovat opiskelijoiden olleen motivoituneita. Oppilaat jaettiin algoritmin perusteella neljään tasaiseen 10 hengen ryhmään. Osa oppilaista jouduttiin kuitenkin

poistamaan kokeesta, eivätkä lopulliset ryhmät olleet tasavahvoja. Oppilaille annettiin yksi luento JUnitin ja TDD:n käytöstä.

Tutkimuksen koehenkilöt tekivät tehtäväviä itsenäisesti. Ryhmät käyttivät eri menetelmiä: kontrolliryhmä ei käyttänyt katselmointia eikä TDD:tä, TDD-ryhmä käytti testivetoista ohjelmistokehitystä, Inspection-ryhmä teki korjauksia tarkistuksen perusteella ja Inspection+TDD-ryhmä käytti molempia menetelmiä. Kolmen tarkistajan ryhmä tarkisti Inspection ja Inspection+TDD ryhmien koehenkilöiden palauttamat tuotokset, jonka jälkeen oppilaille annettiin viikko aikaa korjata merkittävät viat. Tarkistajat etsivät koodista puuttuvia toimintoja, virheellisiä toimintoja ja virheellistä Java-koodia. Oppilaiden korjattua virheet, he palauttivat raportin jossa tarkistajian havaitsemat virheet oli luokiteltu joko selvitetyiksi, huomioimattomiksi, ei virhettä-luokkaan tai luokkaan muu. Ei virhettä- ja muu-luokkaan luokitelluille virheille vaadittiin myös tarkempi selitys.

Havainnot

Guptan ja Jaloten tutkimusryhmässä ohjelmalla P1 havaittiin koodin laadun olevan parempaa TDD-ryhmässä kuin CCD-ryhmässä. TTD-ryhmän havaittiin käyttäneen vähemmän vaivaa (effort), mutta kehittäjän tuottavuudessa ei löytynyt tilastollisesti merkittävää eroa CCD-ryhmään. CCD-ryhmän käytti vähemmän vaivaa testaamiseen ohjelman kehitysvaiheessa joka saattoi vaikuttaa heikompaan koodin laatuun, toisaalta he myös tarvitsivat enemmän aikaa virheiden havaitsemiseen ja korjaamiseen. Ohjelmalla P2 tulokset olivat tasaisemmat: tutkijat eivät pystyneet kaatamaan nollahypoteesia kehitysmenetelmien vaatiman vaivan yhtäsuuruudesta. Kehittäjien käyttämässä vaivassa tulokset olivat samansuuntaisia kuin ohjelman P1 tapauksessa. CCD-ryhmän ohjelma selvisi paremmin testitapauksien läpäisystä ja koodi oli laadukkaampaa, toisaalta he käyttivät enemmän vaivaa kuin TDD-ryhmä. CCD-ryhmä joutui myös korjaamaan vähemmän virheitä.

Molemmilla ohjelmilla siis havaittiin samankaltaisia tuloksia, tutkijat kuitenkin havaitsivat, että oppilasryhmissä oli eroja: ryhmässä kaksi tehtiin enemmän testausta kuin ryhmässä yksi. Tämä saattaa selittää ohjelmien laatujen eron.

Oppilaiden palauttaman kyselytutkimus selvitti subjektiivisia näkemyksiä menetelmistä. TDD:n testikattavuutta pidettiin luotettavampana, kun taas CDD:n suunnittelua pidettiin parempana. Tämä oli odotettavaa, koska CDD vaatii enemmän ennakkosuunnittelua. Noin 53% oppilaista piti parhaimpana menetelmien sekoitusta, jossa tehtiin ensin suunnitelmia ja sitten käytettiin TDD:tä.

Gupta ja Jalote mukaan TDD-menetelmä oli tehokkaampi, se vaati vähemmän vaivaa ja lisäsi hieman kehittäjän tuottavuutta. Toisaalta koodin laatu oli enemmän sidoksissa testaukseen nähtyyn vaivaan ohjelman kehityk-

sen aikana kuin käytettyyn menetelmään.

Wilkerson, Nunamaker ja Mercer mittasivat tuloksia kahdella mittarilla: jäljellä olevien virheiden määrällä ja kustannuksella. Virheiden määrä laskettiin automaattisten testien virheiden perusteella. Kustannuksella tarkoitetaan käytettyjä miestyötunteja. TDD-ryhmällä laskettiin kehittäjän käyttämät tunnit, ja tarkistustusta käyttävissä ryhmissä myös tarkistajian käyttämä aika.

Inspection+TDD ryhmällä havaittiin vähiten virheitä ja pelkkää TDD:tä käyttävällä eniten, enemmän kuin kontrolliryhmällä. Tulokset ovat kuitenkin vääristyneet johtuen esimerkiksi oppilaiden erilaisista lähtötasoista, joten tuloksia tasoitettiin lähtötasotestien ja tarkistusjärjestyksen perusteella. Tämä nosti TDD:n kolmanneksi ohi kontrolliryhmän. TDD:n kustannukset oli matalimmat, n. 11 tuntia verrattuna kontrolliryhmän 12.79 tuntiin. Tarkistusta ja TDD:tä käyttävän ryhmän kustannus oli 31.71 tuntia kun pelkkää tarkistusta käyttävän ryhmän tulos oli 20.88, tutkimuksen tekijät eivät pystyneet selittämään tätä eroa.

Tutkijat toteavat havainneensa koodin tarkistuksen tehokkaammaksi kuin TDD:n virheiden vähentämisessä, mutta se on myös kalliimpaa. TDD:n vähentää kustannuksia verrattuna kontrolliryhmään, mutta merkittävää eroa virheiden määrässä ei löytynyt.

Lähteet

[1] Gupta, A. ja Jalote, P.: *An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development.* Teoksessa *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, sivut 285–294, Sept 2007.

[2] Wilkerson, J.W., Nunamaker, J.F., Jr. ja Mercer, R.: *Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development.* Software Engineering, IEEE Transactions on, 38(3):547–560, May 2012, ISSN 0098-5589.