Test-Driven Development

Syksy 2006
Jyväskylän yliopisto

Test-Driven Development

- Testilähtöinen ohjelmistojen kehitystapa.
- Tehdään ensin testi, sitten vasta koodi.
- Tarkoituksena ei ole keksiä kaikkia mahdollisia tapoja löytää virheitä, vaan suunnitella ja toteuttaa vaatimukset täyttävä ohjelma.
- TDD ei ole testausta, vaan kehitystä!

Testi ≠ Testi

- Testilähtöisessä ohjelmoinnissa testien avulla suunnitellaan ohjelma.
  - Ensin tehdään testi, sitten vasta ohjelma. Toimitaan valmiiden vaatimusten mukaan, eikä koeteta keksiä uusia samalla.

Testi ≠ Testi

- Yksikkötestauksessa varmistetaan ohjelman toiminta.
  - Testi tehdään ohjelman jälkeen. Kaikki poikkeukset ja erikoisuudet koetetaan löytää ja testata.

Mitä saavutetaan

- Tulos 1: Voidaan olla varmoja siitä, että ohjelma toimii (testien mukaan).
- Tulos 2: Saadaan esimerkkejä ohjelman rajapintojen käytöstä.
- Tulos 3: Ohjelma on ylläpidettävämpi ja testattavampi.

Historia

- Lähtöisin samasta sylttytehtaasta kuin XP (= Extreme Programming), ja Smalltalkista. (Eli Ward Cunninghamilta).
- Mitkään XP:n käytännöt eivät ole uusia (paitsi "tehdään kaikki 110%"), ei tämäkään.
- Nykyään: xUnit työkalut useille kielille.
- Muita nimiä: yksikkötestaus, ohjelmoijatestaus.

TDD säännöt

- Älä kirjoita riviäkään uutta koodia, ennen kuin sinulla on sille epäonnistuva testi kirjoitettuna.
- Poista monikerrat.

Sääntöjen seuraukset

- Suunnittelu on interaktiivista. Ajettava ohjelma antaa palautteen, jonka mukaan edetään.
- Teet itse testisi, sillä ei ole aikaa odottaa toisten kirjoittavan niitä.
- Kehitysympäristön on annettava nopea palaute muutoksista.
- Suunniteltava korkean koheesion ja löyhän kytkennän komponentteja, jotka ovat siten testattavia.

TDD Vaiheet

1. **RED** Kirjoita testi. Aja testi, eikä se mene läpi.
2. **GREEN** Muuta tai tuota ohjelmaa niin kauan, että testin ajo menee läpi.
3. **REFACTOR** Siivoa koodi (eli uudelleenrakenna se).

- Toista ylläolevia, kunnes ohjelma on valmis.

Muita seurauksia

- Laatuosasto (QA) voi tehdä työnsä proaktiivisesti, eikä reaktiivisesti.
- Paremmat arviot, vähemmän yllätyksiä.
- Kommunikointi kehittäjien välillä helpompaa ja nopeampaa (aina toimiva koodi esillä).
- Uusi versio uusin ominaisuuksin tarjolla asiakkaalle joka päivä.

Miksi näin ei tehdä?

- "Tämä on tyhmää ja hidastaa kehitystä!"
- "Tämä estää ajattelun / suunnittelun!"
- "Tämä rikkoo mun flown!"
- Taustalla pelko: "Tämä on liian hankala ominaisuus, enkä tiedä mistä aloittaa!"

Pelko

- Seurauksena epävarmuus, kommunikoimattomuus, ujous ja äreys,
- kun pitäisi opetella, kommunikoida, hakea rohkeasti apua.
- TDD auttaa, se antaa varmuuden siitä, että se mitä on tehty toimii.

Kuinka alkuun?

- Aloita nyt heti, ei kohta tai huomenna.
- Jos et voi aloittaa puhtaalta pöydältä, ala tekemään testejä bugikorjauksille ja uusille ominaisuuksille.
- Joitain asioita on hankalaa testata (GUI, DB), tai erittäin hankalaa testata (salaus, samanaikaisuus).
- Tekemällä oppii! (Harjoitteluun esim. katat.)

Bob-sedän neuvoja

- Bob-setä on "Uncle Bob", eli Robert Martin.
- Kymmenen minuuttia testiajojen välillä on liian pitkä aika.
- Emme korjaa sotkuista koodia, koska pelkäämme rikkovamme jotain. Testilähtöisyys osoittaa heti, jos jotain rikkoontuu.
- (Toimivaa ohjelmaa on eri ikävä testata.)

TDD Vaiheet

1. **RED** Kirjoita testi. Aja testi, eikä se mene läpi.
2. **GREEN** Muuta tai tuota ohjelmaa niin kauan, että testin ajo menee läpi.
3. **REFACTOR** Siivoa koodi (eli uudelleenrakenna se).

- Toista ylläolevia, kunnes ohjelma on valmis.

Kirjoita testi

- Kirjoita testi selkeästi.
- Kirjoita se, kuten kirjoittaisit esimerkkiä (testattavan) ominaisuuden käytöstä.
- UB: Kirjoita vain sen verran, että testi ei mene läpi.

Muokkaa, kunnes toimii

- Yksinkertaisin koodi, millä testi toteutuu!
- Älä pelkää tyhmiä ratkaisuja tai järjetöntä koodia. Koodin on oltava yksinkertaista.
- Liian hienoa kikkailua tulee välttää.
- UB: Älä kirjoita mitään, mille ei ole testiä.
- UB: Kirjoita vain sen verran, että testi menee läpi.

Siivoa ohjelmakoodi

- Poista turhat kikkailut ja tuplakoodi.
- Tuplakoodin huomaaminen voi olla hankalaa.
- Siisti myös testit!

Variaatio

- Kirjoitetaan yksi rivi testistä.
- Ajetaan testi, muokataan koodia, kunnes testi menee läpi.
- Kirjoitetaan seuraava rivi testistä.
- ... toistetaan, kunnes valmis.

JUnit = Java + TDD

- Jos JUnitia ei löydy kehitysympäristöstäsi, vaihda kehitysympäristöä.
- Luokasta `junit.framework.TestCase` periytetään omat testiluokat.
- Nimeksi jotain `Test`-alkuista, vaikka `TestJäsen`.
- Java 1.5:ssä hieman hienompi toteutus. (HT)

JUnit testit

- Testit test-alkuisiin, julkisiin ja arvoa palauttamattomiin metodeihin, esim.
```
public void testLisääJäsen() { /*…*/ }
```
- Testaus TestCase-luokan assert-metodeilla, esim.
```
assertEquals("Aku", jäsen.etunimi());
```

JUnit testin ajo

- Testien ajo palauttaa vihreän palkin, jos testit menivät läpi.
- Jos eivät, tulee punainen palkki ja virheilmoitus, joka kertoo virheen paikan.
- Assert-metodeille voi myös antaa kolmantena parametrina merkkijonon, joka selittää testin. Se tulostetaan virheilmoituksen yhteydessä.

JUnit apuja

- Jokainen test-alkuinen metodi suoritetaan erikseen.
- Metodi `setUp` suoritetaan ennen jokaista test-alkuisen metodin ajoa.
- Metodi `tearDown` suoritetaan jokaisen test-alkuisen metodin ajon jälkeen.
- Konstruktori suoritetaan vain kerran.

Lyhyesti

- Testi ensin, sitten koodi. Muista siivota.
- Testilähtöinen kehitys ei ole sama asia kuin perinteinen V-mallin yksikkötestaus.
  - Kehitetään, ei etsitä virheitä.
  - Edustava ote testeistä riittää.
- "Eat your own dog food!"

Linkit

- JUnit & TDD: http://www.junit.org/
- Bob-sedän pakinoita: http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

Viikkotehtävä

- Keilailu-kata: http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata

1. Tee keilailukata.
2. Tee keilailukata.
3. Tee keilailukata.
4. Tee keilailukata.
5. Tee keilailukata.
6. Bonustehtävä: Toista edelliset tehtävät jollain toisella ohjelmointikielellä.

Voit myös heti aluksi tehdä tehtävät jollain sinulle uudella ohjelmointikielellä!