Poglavlje 2

Degenerisani objekti

Opšti TDD ciklus ide ovako.

1. Napišite test. Razmislite kako biste želeli da se operacija iz vašeg uma pojavi u vašem kodu. Pišete priču. Izmislite interfejs koji biste želeli da imate. Uključite sve elemente u priču za koje zamišljate da će biti neophodni za izračunavanje tačnih odgovora.
2. Pokrenite ga. Brzo dovođenje te trake u zeleno dominira nad svime ostalim. Ako je čisto, jednostavno rešenje očigledno, onda ga ukucajte. Ako je čisto, jednostavno rešenje očigledno, ali će vam trebati minut, onda to zabeležite i vratite se na glavni problem, a to je da traka pozeleni za nekoliko sekundi. Ova promena u estetici je teška za neke iskusne softverske inženjere. Oni znaju samo kako da se pridržavaju pravila dobrog inženjeringa. Brzo zeleno opravdava sve grehe. Ali samo na trenutak.
3. Ispravite ga. Sada kada se sistem ponaša, ostavite grešne načine nedavne prošlosti iza sebe. Vratite se na pravi i uski put softverske pravednosti. Uklonite dupliranje koje ste uveli i brzo dođite do zelene boje.

Cilj je čist, jasan kod koji radi (hvala Ronu Jeffriesu na ovom jezgrovitom rezimeu). Čist kod koji radi je ponekad izvan dometa čak i najboljih programera, a većinu vremena izvan dometa većine programera (poput mene). Podeli pa vladaj. Prvo ćemo rešiti deo problema "koji radi". Zatim ćemo rešiti deo "čist kod". Ovo je suprotno od razvoja vođenog arhitekturom, gde prvo rešavate "čist kod", a zatim se mučite pokušavajući da integrišete u dizajn stvari koje naučite dok rešavate problem "koji radi".

$5 + 10 CHF = $10 ako je kurs 2:1
~~$5 * 2 = $10~~
Neka "amount" bude privatno
Sporedni efekti dolara?
Zaokruživanje valute?

Uspeli smo da nateramo jedan test da proradi, ali smo pritom primetili nešto čudno: kada izvršimo operaciju na `Dollar` objektu, taj `Dollar` se menja. Želim da mogu da napišem:

```
public void testMultiplication() {
   Dollar five= new Dollar(5);
   five.times(2);
   assertEquals(10, five.amount);
   five.times(3);
   assertEquals(15, five.amount);
}
```

Ne mogu da zamislim čist način da se ovaj test natera da radi. Posle prvog poziva `times()`, pet više nije pet, već je zaista deset. Međutim, ako vratimo novi objekat iz `times()`, onda možemo množiti naših originalnih pet dolara ceo dan i nikada se neće promeniti. Menjamo interfejs `Dollar` objekta kada pravimo ovu promenu, tako da moramo promeniti test. To je u redu. Naše pretpostavke o pravom interfejsu verovatno nisu ništa bolje od naših pretpostavki o pravoj implementaciji.

```
public void testMultiplication() {
   Dollar five= new Dollar(5);
   Dollar product= five.times(2);
   assertEquals(10, product.amount);
   product= five.times(3);
   assertEquals(15, product.amount);
}
```

Novi test se neće kompajlirati dok ne promenimo deklaraciju `Dollar.times()`:

Dollar

```
Dollar times(int multiplier) {
   amount *= multiplier;
   return null;
}
```

Sada se test kompajlira, ali ne radi. Napredak! Da bi proradio, moramo vratiti novi `Dollar` sa ispravnim iznosom:

Dollar

```
Dollar times(int multiplier) {
    return new Dollar(amount * multiplier);
}
```

$5 + 10 CHF = $10 ako je kurs 2:1
~~$5 * 2 = $10~~
Neka "amount" bude privatno
~~Sporedni efekti dolara?~~
Zaokruživanje valute?

U Poglavlju 1, kada smo naterali test da proradi, počeli smo sa lažnom implementacijom i postepeno je učinili stvarnom. Ovde smo ukucali ono što smo smatrali ispravnom implementacijom i molili se dok su testovi radili (kratke molitve, naravno, jer pokretanje testa traje nekoliko milisekundi). Pošto smo imali sreće i test je proradio, možemo precrtati još jednu stavku.

Slede dve od tri strategije koje poznajem za brzo postizanje "zelenog":

- Fingiraj - Vrati konstantu i postepeno zamenjuj konstante varijablama dok ne dobiješ pravi kod.
- Koristi očiglednu implementaciju - Ukucaj pravu implementaciju.

Kada koristim TDD u praksi, često se prebacujem između ova dva načina implementacije. Kada sve ide glatko i znam šta da ukucam, ubacujem očiglednu implementaciju za očiglednom implementacijom (pokrećući testove svaki put da bih osigurao da je ono što je meni očigledno i dalje očigledno računaru). Ako dobijem neočekivanu crvenu traku, vraćam se nazad, prelazim na fingiranje implementacija i refaktorišem na pravi kod. Kada mi se vrati samopouzdanje, vraćam se na očigledne implementacije.

Postoji i treći stil TDD-a, Triangulacija, koji ćemo demonstrirati u Poglavlju 3. Međutim, da se podsetimo, mi smo

- Preveli primedbu na dizajn (sporedni efekti) u testni slučaj koji je pao zbog te primedbe
- Učinili da se kod brzo kompajlira sa "stub" implementacijom
- Učinili da test radi ukucavajući ono što se činilo ispravnim kodom

Prevođenje osećaja (na primer, neprijatnost zbog sporednih efekata) u test (na primer, dvostruko množenje istog `Dollar` objekta) česta je tema TDD-a. Što duže ovo radim, to sam sposobniji da prevedem svoje estetske procene u testove. Kada to mogu, moje diskusije o dizajnu postaju mnogo zanimljivije. Prvo možemo govoriti o tome da li sistem treba da radi *ovako* ili *onako*. Kada se odlučimo za ispravno ponašanje, možemo govoriti o najboljem načinu postizanja tog ponašanja. Možemo nagađati o istini i lepoti koliko god želimo uz pivo, ali dok programiramo, možemo ostaviti uzvišene rasprave iza sebe i pričati o konkretnim slučajevima.