**Szoftvertechnológia beadandó esszé**

**Test Driven Development módszertan bemutatása.**

**Készítette: Márki Zoltán (WOS3KO)**

**z.marki85@gmail.com**

Neumann jános Egyetem

GAMF

Mérnökinformatikus (levelező) szak BSc

Szoftvertechnológia (GINFBAL-SZOFTTEC-1) tárgy

2020/2021 1. félév

Bevezetés

Szoftverfejlesztés során szükségünk van tesztelésre, azért hogy a szoftver üzembehelyezése előtt megtaláljuk a hibákat és ezzel minőségibb és megbízhatóbb terméket hozzunk létre. Elméletem szerint az az ember, aki még nem esett el biciklivel az még nem is biciklizett igazán, ebből kiindulva, ha fejlesztünk, akkor is fogunk hibázni. Abban igy biztosak vagyunk, hogy tesztelés elött van hiba, de hogy tesztelés után nem marad, azt már nem tudjuk állitani maximum a letesztelt részben nem lesz.

A tesztek írására szinte azonnal lett igény, hiszen egyszerűbb egy automatizált tesztelést végrehajtani, mint manuálisan tesztelni, ami még költségesebb is.

Hagyományos tesztelés úgy működik, hogy a már kész szoftverre sokszor egy külön tesztelő csapat készit. Ez rengeteg erőforrást köthet le mind emberi mind anyagi részről. Az esszémben bemutatok egy másik módszertant. Ez nem más, mint a teszt vezérelt fejlesztés (TDD).

A teszt vezérelt fejlesztés segítségével szoftverek minőségének biztosítása a fejlesztési folyamat részévé válik, ugyanakkor biztonságos alapot teremt a szoftver jövőbeli változtatásához.

Mi az a TDD?

Az ötlet a következő, mi lenne, ha lenne egy eszköz, ami folyamatosan ellenőrizné a fejlesztő munkáját és jelezné, ha hiba történik? Ezzel meg lehetne spórolni rengeteg időt, energiát és költséget.

Mára már létezik ez, ugyan nem egy eszköz, hanem egy módszertan, Test Driven Develpoment röviden TDD. A TDD teszt írás-kódolás-refaktorlálás hármas ciklusát jelenti. A fejlesztés során ez a három lépés követi egymást addig amig a szoftver el nem készül. A teszt vezérelt fejlesztés nem egyszerű, kicsit más szemmel kell a feladatra nézni a megszokottnál, de ez tanulható. Ezt használva jelentősen javítható a kód minősége és könnyebbé teszi a dokumentálást, amit a fejlesztők szintén nem szeretnek.

Működése

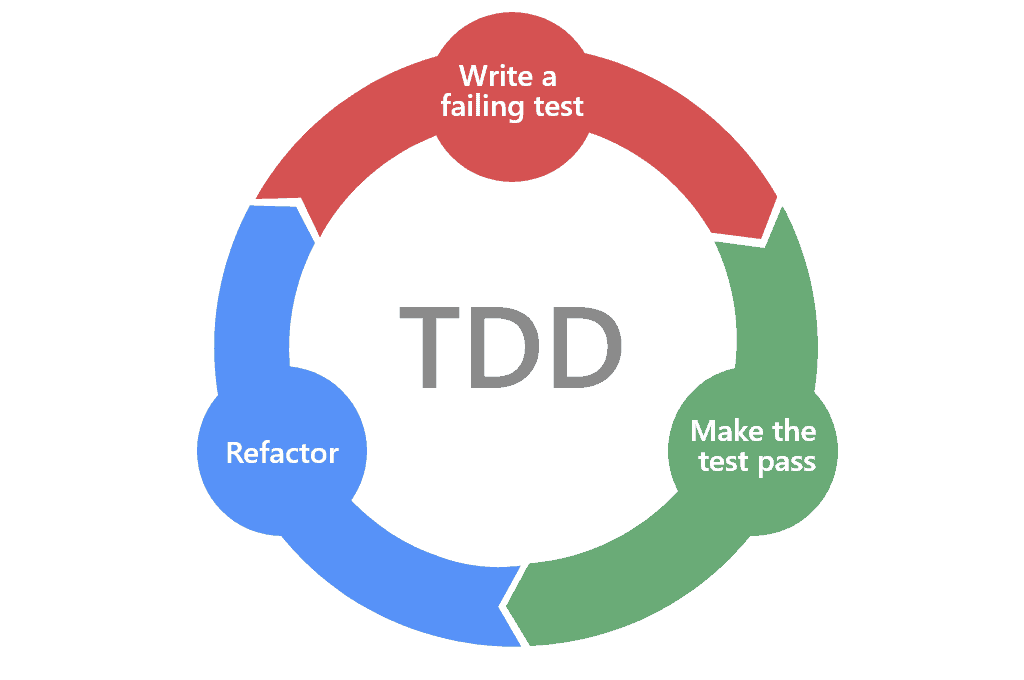
A fejlesztés során először a tesztet írjuk, ami elsőre kicsit szokatlan lehet, de végig gondolva találunk benne logikát. A tesztben konkrétan meg kell fogalmaznunk, hogy mit, szeretnénk elérni, milyen osztályokra, metódusokra van szükségünk és azokat hogyan szeretnénk használni. Ekkor egy „interfészt” hozunk létre, tehát megadjuk, hogy lenne kényelmes a programrészt használni. Ha ezt megírtuk, akkor a teszt sikertelenül fog lefutni, hiszen a jelenlegi állapotban nincs még mögötte kód.

Ekkor neki kezdhetünk a kódolásnak, ahol a feladatunk, hogy a korábban megírt teszt sikeresen lefusson.

Mindig az elvárt működésre írunk tesztet és az osztályok publikus interfészeit vizsgáljuk. Ha azt tesztelnénk, hogy az osztályunk milyen metódusokat, osztályokat milyen sorrendben hív meg akkor később nehezen tudnánk változtatni.

Ha betartjuk a TDD szabályokat, akkor a szoftver minden állapotára készül teszt, így 100%os lesz a kódlefedettség. Ennek a nagy lefedettségnek köszönhetően, ha a kódot úgy módosítjuk ami nem várt módon befolyásolja a szoftvert akkor azonnal kapunk visszajelzést, mert elbukik valahol a teszt.

Folyamata

A teszt vezérelt fejlesztés három fázisból áll, amit három színnel szoktunk jelölni (piros, zöld, kék). Az első fázis a tesztírás (piros), az implementációból és a kód tökéletesítésé azaz refaktorálás.

A fejlesztés során mind a három lépést a lehető legkisebb és leggyorsabb módosítással igyekezzünk elvégezni. Ezt megfelelő gyakorlást követően könnyedén teljesíteni tudjuk. A refaktorálás végezetével újra az első fázis következik ezt a ciklust addig folytatjuk amig a programrészünk el nem készül.

Ezt a tesztelést ugy szokták hatékonyabbá tenni hogy párban dolgoznak, amig valaki foglalkozik azzal hogy kielégitse a tesztet, addig a másik feljelsztő már gondolkodhat és készítheti a következő tesztet.

A következökben részletezném az egyes fázisokban mi történik.

* Piros fázis:  
   Első lépés ugye a teszt írása arra a viselkedésre amit el szeretnénk majd érni. Ekkor vagyunk a piros fázisban, ami azért kapta ezt a megnevezést mert itt egy olyan tesztet kell írni amit tudunk hogy el fog bukni.  
  Ebben a részben tulajdonképpen megtervezzük a viselkedéshez szükséges interfészt, azt hogy milyen osztályra milyen metodusra van szükségünk. A TDD-nek köszönhetően rá vagyunk kényszeritve hogy inkább osztályt használó szemszögből közelitsünk meg a kódunkat ezáltal sokkal felhasználóbarátabb lesz.  
  Az ideális teszt nem túl hosszú, törekedni kell arra hogy 4-5 sor legyen, de ez nem mindig kivitelezhető az elő- és utóteszteknél. Ilyenkot a további teszteket egyszerübben meglehessen majd irni. Ha nem sikerül egyszerüen megirni a teszt akkor szinte biztos hogy túl sokmindent akarunk egyszerre tesztelni. Ilyenkor végig kell gondolni hogyan tudnánk jobban szétvágni az adott problémát több részre.  
  Ha készen vagyunk a teszt megirásával akkor minden esetbe futassuk le, de ne csak az új részt hanem az egészet és gyözödjünk meg róla hogy csak a legutolsó fog elbukni. Ha véletlenül nem bukik el vagy nem azt a hibát generálja amire számitottunk akkor valamit rosszul terveztünk vagy már esetleg hoztunk létre egy ilyen osztályt.
* Zöld fázis:  
  A következő fázis az implementáció írása, itt is törekedni kell arra hogy a legrövidebb kóddal elérjük azt hogy a teszt sikeresen fusson le. Itt még nem fontos az, hogy elegáns legyen a megoldás. csak az a fontos hogy a teszt zöld legyen. Soha nem szabad többet megvalósítani, mint amit a teszt elvár, ha bármi mást megvalósítaná akkor arra már másik teszt kéne. Igy tudjuk elérni a 100%-os lefedettséget.  
  Ha elkészült a kód, akkor futtassuk le az összes tesztet, és ha tényleg mindent jól csináltunk, akkor zöld utat kaptunk a következő lépésre.  
  Ha véletlenül túl sok időbe telik a kódírás, akkor át kell gondolni, hogy pontosan mit szerettünk, volna csinálni. Próbáljuk a megírt tesztet kisebbre bontani. Lehet hogy egy komplexebb viselkedést átgondolva újabb osztályok esnek ki. Ilyenkor próbáljuk először ezeket a segéd osztályokat megírni, persze előtte tesztet kell rá készíteni.
* Refaktorálás:  
  Az utolsó rész a kód minőségének a javítása. Az előző részben gyorsan megoldottuk, hogy működjön a feladat, most az lesz a feladatunk, hogy átláthatóbbá és elegánsabbá varázsoljuk. Itt most nem kell sietni, itt az a lényeg, hogy megfelelő legyen a kód külalakja, a karbantarthatóság miatt átlátható legyen, de fontos hogy eközben soha nem változtathatunk az alkalmazás viselkedésén.  
  Refaktorálás közben lehetőségünk van már átgondolni, hogy a következő tesztnél mire lesz szükségünk.  
  Minden módosítás után futtassuk le a teljes tesztet, igy ellenőrizhetjük, hogy nem rontottunk el semmit. Ilyenkor már szabadon alakítható kód, hiszen a tesztek biztosítják a szoftver helyes működését, ha elbukik akkor tudjuk hogy véletlenül olyat módosítottunk amit nem kellett volna.  
  Nem szabad megfeledkezni arról, hogy a tesztünket is refaktorlálni kell. Hiszen ha ismétléseket hagyunk benne, vagy komplex sok soros tesztet hagyunk akkor később nehezen lesz áttekinthető és bővíteni. A tesztek minősége azért is fontos mivel a tesztkészletünk mindennél jobban dokumentálja a szoftver működését. A TDD mellett nem feltétlenül fontos a fejlesztői dokumentáció.

A refaktorálást követően a ciklus kezdődik elölről addig amig a szoftver el nem készül. Nem szabad elfelejteni, hogy a TDD kulcsa a minél apróbb kis lépésekben haladás, hiszen annál rugalmasabban tudjuk kezelni az esetleges nem várt problémákat.