Test driven development (TDD)

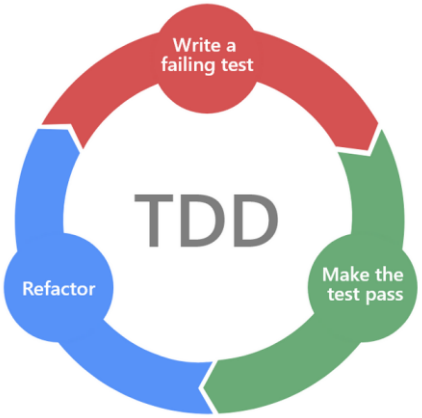
Prof. univ. dr. ing. Florica Moldoveanu

Curs *Ingineria programelor* – UPB, Automatică şi Calculatoare
2020-2021

Ce este TDD?

❖ Este o practică în dezvoltarea de software, cu originile în metodologia agilă XP (*Extreme programming),* folosită în prezent nu numai în contextul unei dezvoltări agile.

Test driven development = Test first development + Refactoring



[2]

Test-first development:

- *Programatorul scrie testele unitare înainte de a scrie codul testat*
- Testele sunt dezvoltate incremental pornind de la scenarii.
- Fiecare test corespunde unei cerințe.
- Reprezentantul utilizatorilor/clientului este implicat în dezvoltarea şi validarea testelor.
- Sunt folosite instrumente de testare unitară automată.
- Atunci când este adăugat/modificat cod sunt rulate toate testele, atât cele anterioare cât şi noul test – se verifică astfel ca noul cod nu a introdus erori.

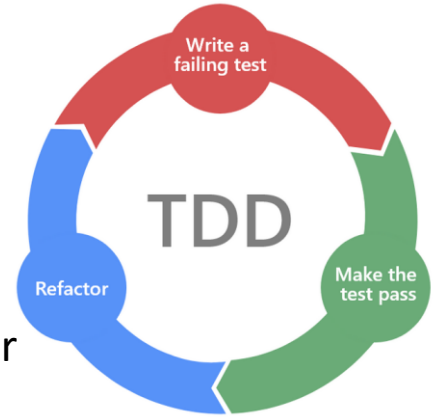
Ce este TDD?

Refactorizare (îmbunătățirea codului)

▪ *Rescrierea codului, fără a modifica funcționalitatea sa externă, pentru îmbunătățirea unor atribute ale codului, cum ar fi: eficiența, claritatea, complexitatea, uşurința de întreținere.*

➢ Codul este mai uşor de înțeles, reduce necesitatea documentației.

➢ Modificările se fac mai uşor deoarece codul este bine structurat şi clar.



[2]

Rosu.

- Se scrie un test unitar, înainte de a scrie codul testat. Rezultatul execuției testului este o cădere (failure), caci codul testat nu exista - eroarea de compilare este o cădere.

Verde.

- Se scrie într-un timp scurt codul care trebuie să satisfacă testul, chiar dacă nu este o soluție de implementare bună. Rezultatul execuției testului trebuie sa fie “success” (passed).

Albastru

- Refactorizarea se aplică atât codului testat cât şi codului de test, şi nu trebuie să modifice comportarea externă a programului.

Se reexecuta toate testele. Se fac ajustări până când toate testele trec.

TDD – paşii (1)

Presupunem că noua funcționalitate se va implementa printr-o **functie f, dintr-o clasă C**, şi folosim Junit pentru testarea unitară.

Pentru testarea funcției f din **clasa C** se implementeaza o functie **test_f**, în **clasa de test CT**, parcurgându-se următorii paşi:

repetă

scrie un test în functia test_f, în care se apelează funcția **f**, folosind o aserțiune

repetă

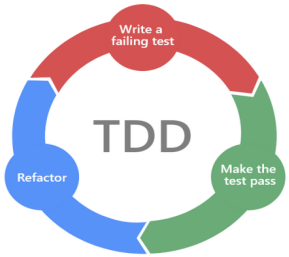
- rulează programul de test (cu toate testele definite în functia test_f)
- dacă rezultatul rulării este “failed” **adauga (după prima rulare)/modifică funcția f**

până când rezultatul rulării este “passed”

repetă

- refactorizează codul funcției **f** şi al funcției **test_f**
- rulează programul de test (cu toate testele definite în functia **test_f**)

până când toate testele definite în **test_f** trec (nici un test nu a produs cadere)



până când nu mai sunt teste noi pentru funcția f

TDD – caracteristici

- Se acordă aceeaşi importanță codului de test ca şi codului testat.
- **Testele trebuie să fie clare: uşor de înțeles**
- **Un (caz de) test trebuie să corespundă unei cerinte şi să conțină o singură aserțiune**.
- Testele trebuie sa fie independente intre ele.
- **Testele trebuie sa fie repetabile: reproductibile în orice context.**
- Fiecare test trebuie să aibă un rezultat binar: cădere sau success.
- Un test trebuie să fie scris imediat înainte de scrierea codului pe care-l validează.
- **Testele reprezintă specificații executabile şi fac parte din documentația aplicației.**
- Aplicarea TDD necesită o perioadă de învățare - este o shimbare de paradigmă – dar este o investiție în viitor → conduce la creşterea productivității activității de dezvoltare.
- Aplicarea TDD necesită instrumente bune de testare unitară automată.

TDD – beneficii (1)

- **Conduce la dezvoltarea de software cu grad înalt de uzabilitate (usability):** uşurința de înțelegere, de învățare, de operare, plăcerea de a utiliza produsul, de către utilizatorii țintă

  - abordarea TDD îi forțează pe dezvoltatori să se focalizeze pe problemele de uzabilitate pe tot parcursul dezvoltarii;

  - uzabilitatea este asigurată prin testele unitare: reprezentantul utilizatorilor este implicat în dezvoltarea şi validarea testelor.

- **Reduce numărul de defecte din cod:** sunt eliminate in mare parte prin tesele unitare
- **Evidențiază problemele de funcționalitate în fazele incipiente** → reduce costul rezolvarii problemelor de implementare a funcționalitatilor cerute.
- **Simplifică codul**: solicită dezvoltatorului să scrie cod care trebuie să raspundă cerințelor de testare; se evită scrierea de cod care nu este necesar.

TDD – beneficii (2)

- **Conduce la cod extensibil**: dezvoltatorul este încurajat să construiască software alcătuit din unități mici de cod testabil.
- **Permite obtinerea de feedback rapid de la client/utilizatori.**
- **Documentarea actualizată a codului: testele sunt exemple de utilizare a codului.**
- Scurtează timpul de dezvoltare: abordarea “test-and-code” creşte productivitatea echipei de dezvoltare. Permite scrierea de cod funcțional într-un timp mai scurt decât alte practici de dezvoltare.

Resurse

1. https://hackernoon.com/introduction-to-test-driven-development-tdd-61a13bc92d92
2. https://marsner.com/blog/why-test-driven-development-tdd
3. Kent Beck, Test Driven Development: By Example, Addison Wesley, 2003
4. https://www.infoq.com/articles/test-driven-design-java/
5. Un demo cu aplicarea TDD în testarea unitară: *https://www.youtube.com/watch?v=ZHZY4lVm6co&feature=emb_title*