Szoftvertechnológia

Teszt vezérelt szoftverfejlesztés

Clean code

Dr. Szendrei Rudolf

ELTE Informatikai Kar

2020.

Teszt vezérelt szoftverfejlesztés

Teszt vezérelt szoftverfejlesztés

Definíció

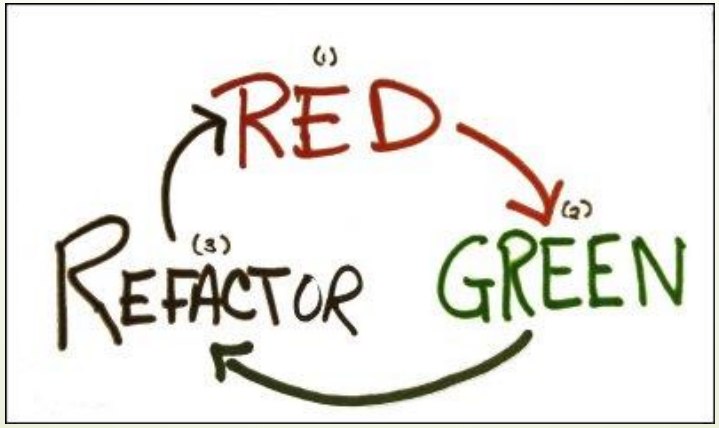
- A TDD a szoftverfejlesztés egy evolúció alapú megközelítése, amely ötvöződik az „előbb-teszteljünk” fejlesztési módszertannal.
  A teszteket már azelőtt megírjuk, mielőtt elkészítenénk azt az éppen elegendő kódot, ami átmegy a teszten, majd ezt refaktoráljuk.

- Ez is egy módja annak, hogy átgondoljuk a követelményeket vagy a tervet, mielőtt megírnánk a működő kódot.

Uncle Bob törvénye (Robert C. Martin)

A TDD három alapszabálya

- NE kódoljunk semmit, kivéve ami ahhoz kell, hogy a programunk átmenjen a sikertelen teszten.
- Tesztből csak éppen elegendő mértékűt írjunk a hiba demonstrálásához.
- Csak annyi kódot írjunk, amennyi éppen elegendő a sikeres teszthez.

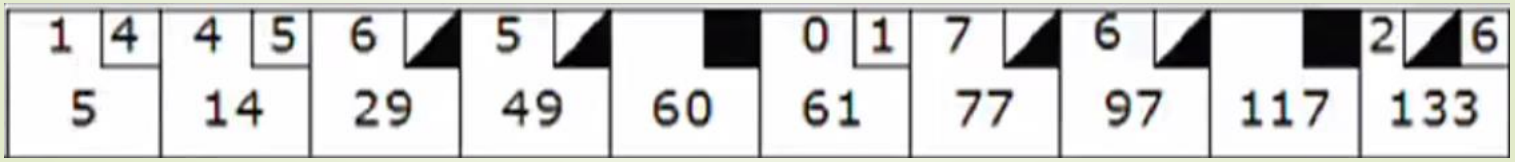


+1 szabály

- Legyünk óvatósak az előbbi szabályokkal!
- Azt mondják általában, hogy csak annyit kódoljunk, ami pillanatnyilag épp szükséges. Ez nem elég!
- Végeredményként egy tökéletesen struktúrált és refaktorált kódot kell kapnunk, ami nem csak a teszteket elégíti ki.
- **A tesztek a lehetséges használati eseteknek csak egy részhalmazát fedik le.**
- A végső kódnak jól kell működnie az összes lehetséges tesztesetben. Ne felejtsük ezt el!

Bowling példa

- Készítsünk egy programot, ami kiszámítja a Bowling játék során szerzett pontunkat.
- Tíz mezőt kell teljesítenünk, és minden mezőben két gurítási lehetőségünk van.
- Ha tarolunk (strike), a következő két gurítás eredménye hozzáadódik a tarolás eredményéhez.
- Ha másodikra döntjük le a bábukat (spare), akkor csak a következő gurítás eredménye adódik pluszban a pontszámokhoz.
- A 10. mező hibátlan teljesítésénél bónusz dobást kapunk (strike-nál kettőt, spare-nél egyet).



Clean Code

Mi a Clean Code

- Olvasható
- Karbantartható
- Tesztelhető
- Elegáns

Ezek mind szubjektív tulajdonságok

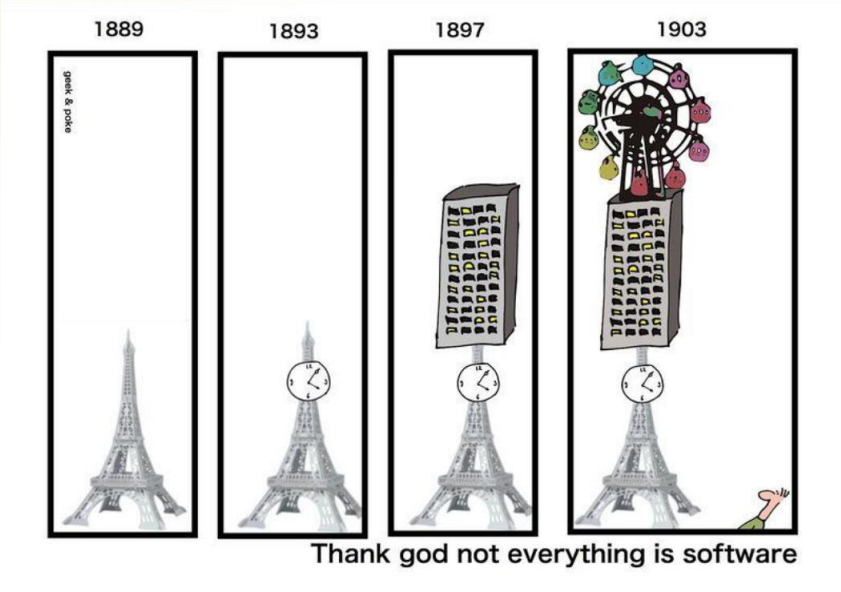
Miért fontos?

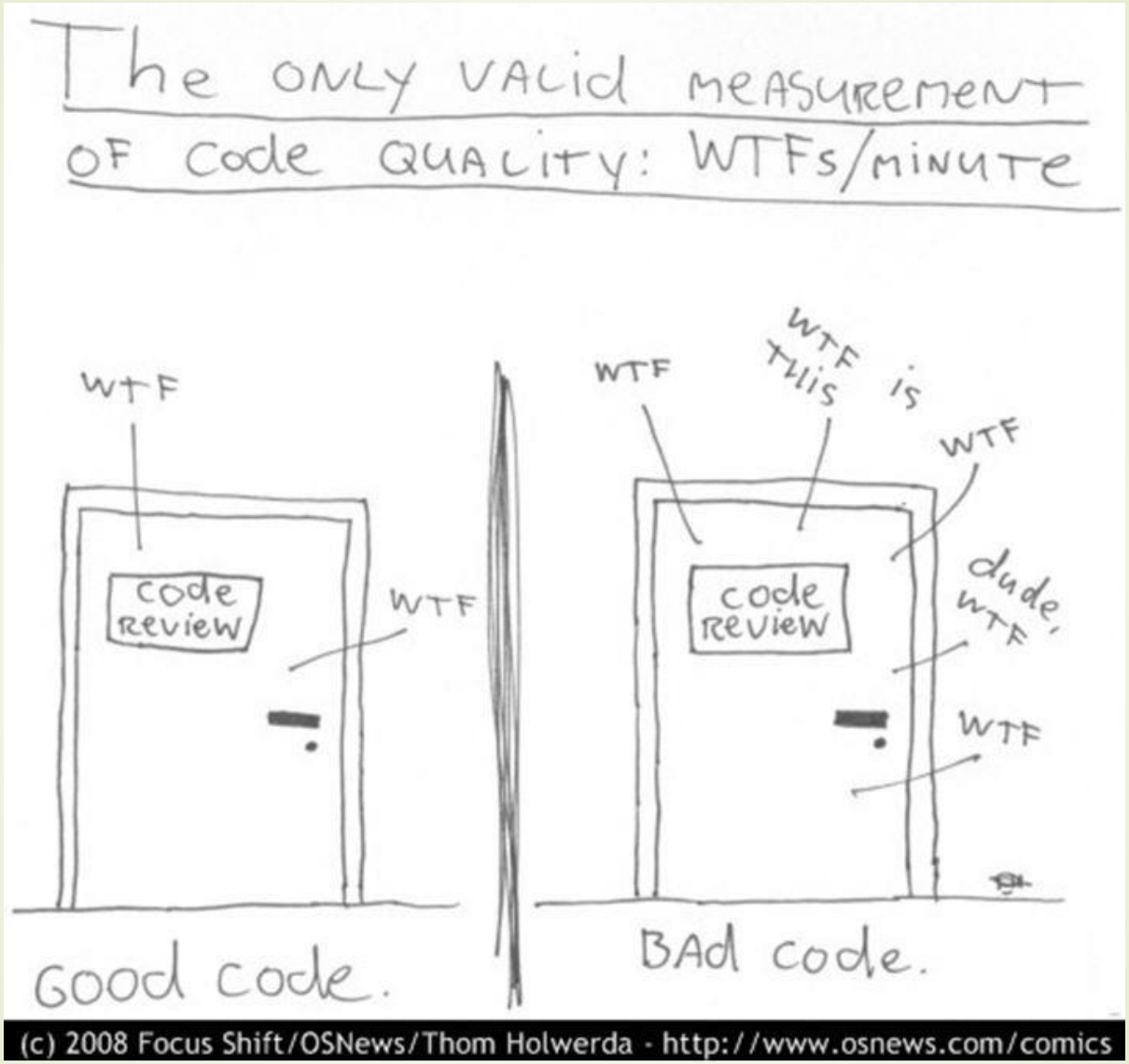
A megírt kód folyton változik

- Hibajavítások
- Követelmény változások
- Új funkciók

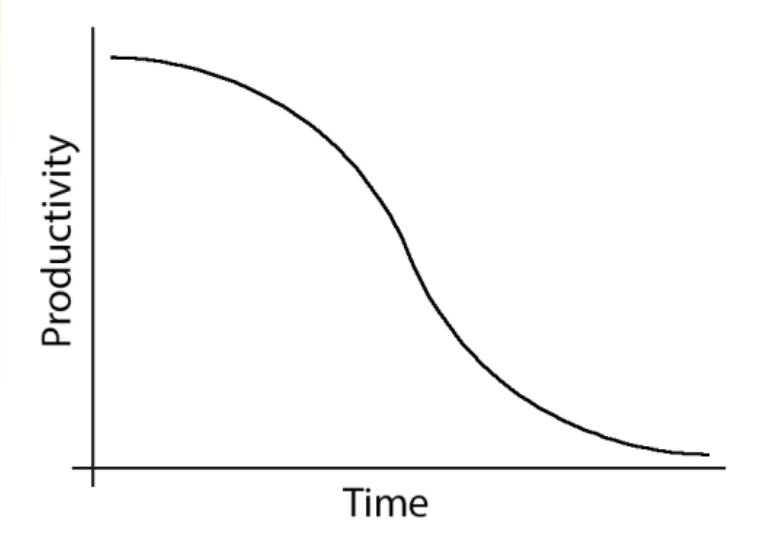
Tiszta kód hiányában

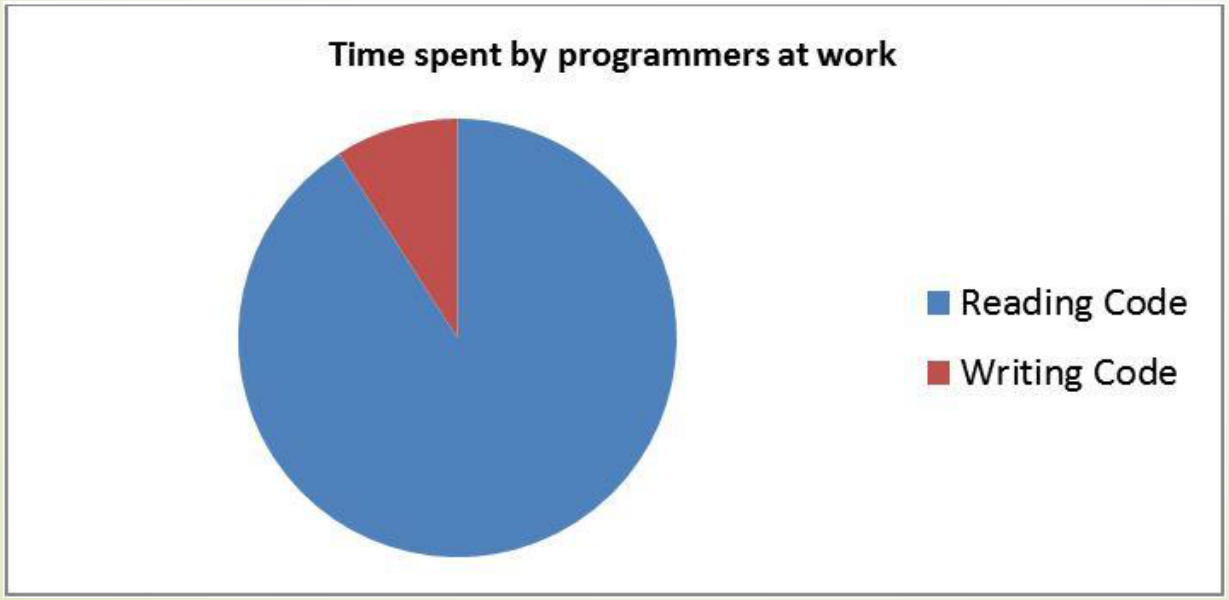
- A fejlesztési idő megnő
- Nehezen felderíthető hiábák




The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF this is
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

- A kód minősége közvetlenül hatással van a karbantarthatóságra.





Clean code probléma

- Majd megcsinálom később:
  - Később soha nem jön el.
- A kódot egyszerűbb tisztán tartani, mint később rendbetenni.

Alapelvek

Elnevezések

„Choosing good names takes time
but saves more than it takes.”

Use Intention-Revealing Names

- int d; // elapsed time in days

Mit fejez ki d ? Semmit. Sem az eltelt időre, sem a napokra nem vonatkozik.

- int elapsedTimeInDays;

Félrevezető nevek

- theList - Not very good
- ProductList - A bit better
- ProductCatalog – Good

Kiejthető / Kereshető nevek

- Date genymdhms; VAGY Date generationTimestamp;
- Konstansok, előfordulások keresése?
- MAX_ORDER_BY_CUSTOMER vs. 6

Elnevezések

- Egy szó per koncepció:
  - Összezavaró ha ugyanarra a dologra több névvel hivatkozunk. Pl.:
  - fetch, retrieve, get mint ekvivalens metódusok különböző osztályokban.
- Ugyanazt a nevet ne használjük különböző célra
- Kerülendőek a név prefixek. Pl.:
  - Project név: „My Project”
  - MPOrderService

Függvények, metódusok

- Rövid: maximum egy képernyőre rá kell férnie
- Don’t Repeat Yourself (DRY) - copy/paste = bad
- Egyetlen dolgot csinál – Egy metódus, egy absztrakciós szint
  Magasabb szintől → részletek
- Blokkoknak egyértelmű be- kilépési pont, ~~break, continue~~

```
public Money calculatePay(Employee e)
throws InvalidEmployeeType {
  switch (e.type) {
    case COMMISSIONED:
      return
calculateCommissionedPay(e);
    case HOURLY:
      return calculateHourlyPay(e);
    case SALARIED:
      return calculateSalariedPay(e);
    default:
      throw new
InvalidEmployeeType(e.type);
    }
}
```

- Új employee type esetén a függvény mérete egyre nő
- Sérti a Single responsibility Principle-t
  (több ok miatt is változhat)
- Sérti az Open Closed Principle-t
  (új típus hozzáadása után változtatni kell)
- Más függvényekben is fel kell használni ugyanazt a struktúrát:
  - `isPayday(Employee e, Date date)`
  - `deliverPay(Employee e, Money pay)`
  - ...
- Switch utasítsok elfogadhatóak, ha azok nem ismétlődnek.

Megoldás

```
public abstract class Employee {
  public abstract boolean isPayday();
  public abstract Money calculatePay();
  public abstract void deliverPay(Money pay);
}
public interface EmployeeFactory {
  public Employee
         makeEmployee(EmployeeRecordr);
  throws InvalidEmployeeType;
}
```

```
public class EmployeeFactoryImpl implements
EmployeeFactory {
  public Employee
         makeEmployee(EmployeeRecord r)
  throws InvalidEmployeeType {
    switch (r.type) {
      case COMMISSIONED:
        return new CommissionedEmployee(r);
      case HOURLY:
        return new HourlyEmployee(r);
      case SALARIED:
        return new SalariedEmploye(r);
      default:
        throw new
               InvalidEmployeeType(r.type);
    }
  }
}
```

- Mellékhatások
  - Félrevezetés a függvény feladatával kapcsolatban: a jelzett funkció mellett valami rejtett dolgot is elvégez.

```
public class UserValidator {
  public boolean checkPassword(String userName, String password) {
    User user = UserGateway.findByName(userName);
    if (user != User.NULL) {
      if ("Valid Password".equals(password)) {
        Session.initialize();
        return true;
      }
    }
    return false;
  }
}
```

Command vs query

- Függvények általában vagy végrehajtanak valamit, vagy válaszolnak valamire.
- De nem mindkettőt egyszerre:
  - `public boolean set(String attribute, String value);`
    - Beállítja egy mező értékét és jelzi sikeres volt-e. Probléma?
  - `if (set("username", "tibi"))`
    - Mit jelent ez?
    - Megmondja, hogy a username mező értéke már tibi-re van állítva?
    - Megmondja, hogy sikeres volt-e az érték beállítása?

Hibakezelés

- Hibakódok helyett kivételek jelzik a hibát
- A hibakezelő blokkok tartalmát célszerű metódusokba kirakni.
- Kerülendő null érték visszaadása metódusokból

```
List<Employee> employees = getEmployees();
if (employees != null) {
    for (Employee e : employees) {
        totalPay += e.getPay();
    }
}
```

- Kerülendő null érték paraméterül adása

```
public double xProjection(Point p1, Point p2) {
    return (p2.x - p1.x) * 1.5; }

public double xProjection(Point p1, Point p2) {
    if (p1 == null || p2 == null) { throw InvalidArgumentException(); }
    return (p2.x - p1.x) * 1.5;
}
```

Kommentek

- A kommentek hazudnak: Kód és komment nem él együtt.
- Komment nem javít a rossz kódon, ha nehezen érthető, át kell írni

Például:

```
// Check to see if the employee is eligible for full
   benefits
if ((employee.flags & HOURLY_FLAG) &&
    (employee.age > 65))
```

VAGY

```
if (isEligibleForFullBenefits(employee))
```

Jó kommentek

- Információs kommentek: Javadoc

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile(
"\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

- Szándékot, pontosítást tartalmazó kommentek:

```
//Sample input: Oct 5, 2015 -13:54:15 PDT
```

- Következményre figyelmeztető kommentek:

```
// We do a deep copy of this collection to make
// sure that updates to one copy do not affect
// the other
```

- TODO, stb.: kommentek
- JavaDoc?

Rossz kommentek

- Zaj – felesleges kommentek:

```
private int counter; // the counter
```

- Kommentek metódusok helyett
- Tagoló kommentek: `// Getters..... ////////////`
- Zárójelek kommetjei:

```
while(...) {...
} // while
```

- Kikommentezett kód
- (HTML comment)

Formázás

- Együtt dolgozó fejlesztőknek célszerű megállapodni a használt formázási szabályokban.
- Mindenkinek ugyanazt a fromázási stílust kell használnia.
- Célja konzisztens kinézetű kód készítése egy terméken belül.

Eszköz ismeret

Always leave the code cleaner than you found it.

- Nem megfelelő információ a változásokról a history-ban.
- `i++; // increment i`
- Kikommentezett kód
- Soha nem használt függvények / kódok
- Duplikátumok
- Szelektor argumentumok (true/false)