Extrémne programovanie a iné agilné metodológie

Zimný semester 2007/08

Ing. František Gyárfáš, PhD.

Katedra aplikovanej informatiky

gyarfas@ii.fmph.uniba.sk

http://www.ii.fmph.uniba.sk/~gyarfas/

Bez čoho XP naozaj neide?

- **Bez párového programovania?**
- **Bez refaktorizácie?** (zjednodušenie kódu bez zmeny funkčnosti)
- **Bez integrácie?** (ihneď po ukončení vývoja)
- **Bez testovania?**

Bez testovania to neide !

Smutná realita testovania

Zákon odkladania na neskoršie

Ak sa testovanie odloží na vtedy, až na to bude čas, softvér otestuje až zákazník.

Čo je to test?

Test v angličtine znamená sloveso **testovať**.

Predpokladám, že pri programovaní testujete všetci. Prečo?

- **Kód, ktorý nebol testovaný, nefunguje.**
- **Kód, ktorý sa nedá testovať, neexistuje.**
- **Programátor, ktorý netestuje svoj kód, je nebezpečný maniakálny lunatik alebo nezodpovedný blázon.**

Problémy s testovaním

Problémy s testovaním:

- **Validnosť testovania zmenou programu zaniká.**
- **Úspešné testovanie nepredvída budúcnosť.**
- **Neskorší vývoj nemusí dať úspešnému testovaniu za pravdu.**

Čo to je test ešte raz?

Test v angličtine znamená aj podstatné meno **test**.

Test je kus spustiteľného programového kódu (funkcia, metóda), ktorý testuje, či nejaká časť programu sa správa tak, ako to od nej očakávame. Či dáva na definovaný vstup očakávaný výstup.

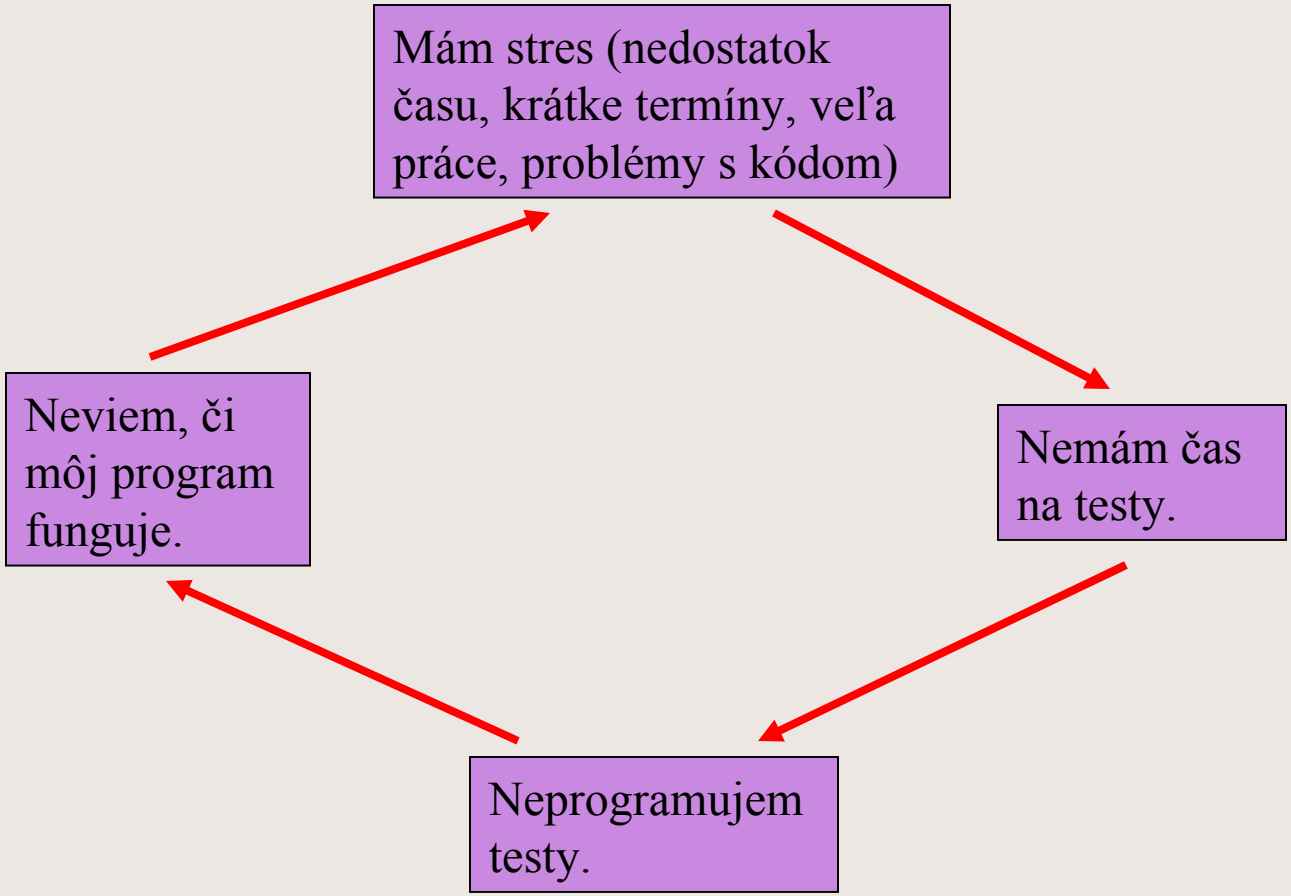
Predpokladám, že takéto testy vytvára priebežne málokto z Vás.

Testy v XP

Testy tvoria chrbtovú kosť Extrémneho programovania. O takýchto testoch sa budeme dnes baviť.

- Test je kus kódu, ktorý volá testovanú funkciu, metódu či triedu, zadá jej úlohu a porovná jej riešenie so správnym riešením.
- Test je možné kedykoľvek počas existencie programu opakovať a vždy musí byť rovnako úspešný.
- Počet testov počas vývoja programu neustále narastá a je potrebné ich neustále všetky zbiehať.

Smrteľná špirála



Zbaviť sa strachu

Pridám kus nového kód, niečo zmením v existujúcom, integrujem dve nezávislé časti kódu a naraz netuším:

- **čo všetko zrazu prestane fungovať,**
- **čo som neúmyselne pokazil,**
- **na čo som zabudol,**
- **v ktorej časti, na ktorú som vôbec nesiahal, môžem čakať zradu.**

Programovanie riadené testmi je cesta, ako sa pri programovaní zbaviť strachu.

Infekcia testmi

Väčšina ľudí, ktorí sa naučili programovať s testmi, zmenili svoje programátorské návyky navždy.

Erich Gamma to nazýva: **infekcia testami**.

Odstránenie infekcie testmi

Na odstránenie infekcie testmi už existuje aj úspešná liečba:

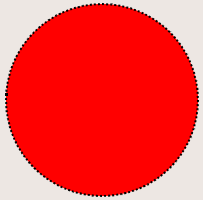
- **stres**
- **nedostatok času**
- **strach zo zlyhania**

Výsledkom úspešnej terapie je zbavenie sa návyku na tvorbu testov.

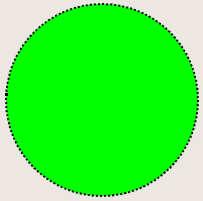
Vyliečení pacienti sa však sťažujú na nedostatok času, hrôzu pred odovzdávaním a veľkú chybovosť.

Semafor

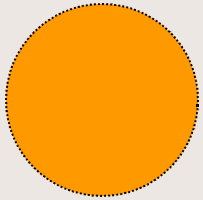
Programovanie s testovaním pripomína semafor.



Napíš test, ktorý nefunguje.



Naprogramuj a oprav kód, aby test fungoval.



Refaktoruj: eliminuj kód, ktorý je duplicitný alebo zbytočný.

Rytmus TDD

Testmi riadené programovanie (TDD) má svoj špecifický rytmus, riadený semafórom:

- Pridaj nový test
- Pusti všetky testy a skontroluj, že nový test zlyhal.

- Urob zmenu programu, ktorá spojazdní zlyhajúci test.
- Pusti všetky testy a skontroluj,či všetky fungujú.
- Ak niektorý nefunguje, oprav program, kým nebudú fungovať.

- Refaktoruj kód, odstráň všetky duplicity.
- Pusti všetky testy a skontroluj, či všetky fungujú.
- Ak niektorý nefunguje, oprav program, kým nebudú fungovať.

- Vráť sa na začiatok.

Kurzová kalkulačka

Firma	Počet akcií	Cena za akciu	Celková cena
IBM	1000	25 USD	25000
HP	400	100 USD	40000
		Suma	65000

Násobenie v tej istej mene.

Sčítanie v tej istej mene.

		kurz
CHF	USD	2

Násobenie v inej mene

Firma	Počet akcií	Cena za akciu	Celková cena
IBM	1000	25 USD	25000 USD
ACER	400	200 CHF	80000 CHF
		Suma	65000 USD

Sčítanie v rôznych menách

1. Úloha - násobenie

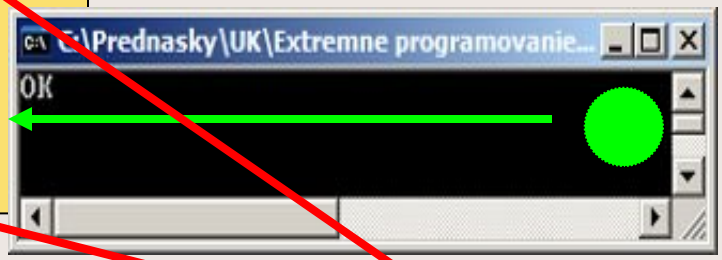
Prvou úlohou je násobenie v tej istej mene.

Napíšeme si najprv test pre násobenie:

```
void testNasobenia(){
    DOLLAR pat(5);
    pat.nasob(2);
    assertEquals(10, pat.hodnota);
}
```

```
class DOLLAR{
  public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    void nasob(int i){hodnota *= i;};
};
```

XP hovorí, že programovať treba iba to najnevyhnutnejšie..

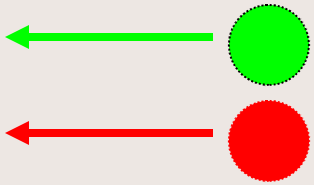



```
Assertion failed: v1 == v2, file C:\Prednasky\UK\Extremne programovanie
\TestDevelopment\TDD_1.cpp, line 24

Abnormal program termination
```

Zdvojený test násobenia

```
void testNasobenia(){
    DOLLAR pat(5);
    pat.nasob(2);
    assertEquals(10, pat.hodnota);
    pat.nasob(3);
    assertEquals(15, pat.hodnota);
}
```



```
class DOLLAR{
  public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    void nasob(int i){hodnota *= i;};
};
```

Násobenie mení hodnotu v samotnej triede.

```
C:\Prednasky\UK\Extremne programovanie\Projekty\TestDevelopment>TDD_1
OK - prvy

Assertion failed: v1 == v2 && test.c_str(), file C:\Prednasky\UK\Extremne progra
movanie\Projekty\TestDevelopment\TDD_1.cpp, line 27

Abnormal program termination
```

Zmenený test násobenia

```
void testNasobenia(){
    DOLLAR pat(5);
    DOLLAR *vysl = pat.nasob(2);
    assertEquals(10, vysl->hodnota);
    vysl = pat.nasob(3);
    assertEquals(15, vysl->hodnota);
}
```

Výsledok násobenia presunieme do premennej `vysledok`.

```
class DOLLAR{
 public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    DOLLAR *nasob(int i){
      return new DOLLAR(hodnota * i);
    };
};
```

Musíme zmeniť aj metódu `nasob`.

Postup k zelenej

Keďže cieľom každého kroku je dostať nový test do zeleného stavu, odporúčajú sa základné stratégie:

- Ak je implementácia jasná, **implementuj**.
- **Fake It** – Ak začínam mať problémy, vložím na príslušné miesto konštantu, ktorú test očakáva a získam znovu zelenú. Až sa ukľudním a nadobudnem znovu sebavedomie, refaktorujem konštantu na správny kód.
- **Triangulácia** – zovšeobecňujeme kód iba v prípade, že máme dve rôzne situácie a druhá si vyžaduje generalizáciu. (Príklad s násobením bol taký.)

Test rovnosti – equals(...)

```
void testRovnosti(){
    DOLLAR pat(5);
    DOLLAR *vysledok = new DOLLAR(5);
    assertTrue(pat.equals(vysledok));


}
```

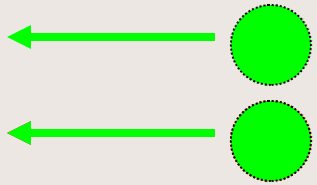
```
class DOLLAR{
  public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    bool equals(DOLLAR *val){
                return true;
    };
};
```

Splnenie testu nás nenúti k skutočnej funkcionalite – **Fake it!**.

Triangulácia - druhý test

```
void testRovnosti(){
    DOLLAR pat(5);
    DOLLAR *vysledok = new DOLLAR(5);
    assertTrue(pat.equals(vysledok));
    vysledok = new DOLLAR(6);
    assertFalse(pat.equals(vysledok));
}
```

Pridáme druhý test – **Triangulácia.**

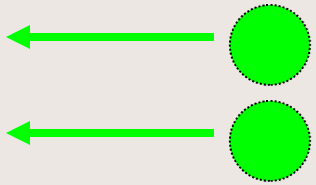


```
class DOLLAR{
  public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    bool equals(DOLLAR *val){
      return (hodnota == val->hodnota);
    };
};
```

Druhý test nefunguje. Je nevyhnutné hľadať riešenie, ktoré splní oba testy.

Prichádzajú franky

```
void testNasobeniaFrankov(){
    FRANK pat(5);
    FRANK *vysledok = pat.nasob(2);
    assertEquals(10, vysledok->hodnota);
    vysledok = pat.nasob(3);
    assertEquals(15, vysledok->hodnota);
}
```



```
class FRANK {
  public:
    int hodnota;
    FRANK(int h){ hodnota = h;};
    FRANK *nasob(int i){
      return new FRANK (hodnota * i);
    };
};
```

Refaktorizácia

```
class DOLLAR{
  public:
    int hodnota;
    DOLLAR(int h){ hodnota = h;};
    DOLLAR *nasob(int i){
      return new DOLLAR(hodnota * i);
    };
};
```

```
class FRANK {
  public:
    int hodnota;
    FRANK(int h){ hodnota = h;};
    FRANK *nasob(int i){
      return new FRANK (hodnota * i);
    };
};
```

Duplicitný kód, ktorý volá po odstránení.

Zovšeobecnenie peňazí

```
class MONEY {
  protected:
    int hodnota;
};


class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){ hodnota = h;};
    DOLLAR *nasob(int i){
       return new DOLLAR(hodnota * i);};
    bool equals(DOLLAR *val){
      return (hodnota == val->hodnota);};
};


class FRANK : public MONEY {
  public:
    FRANK(int h){ hodnota = h;};
    FRANK *nasob(int i){
      return new FRANK (hodnota * i);
    };
    bool equals(FRANK *val){
      return (hodnota == val->hodnota);};
};
```

Chyba.
`hodnota` je `private`, čo neumožňuje prístup k nej.

Zmeníme `hodnota` na `protected`.

Duplicitný `nasob` a `equals`

```
class MONEY {
  protected:
    int hodnota;
};
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){ hodnota = h;};
    DOLLAR *nasob(int i){
       return new DOLLAR(hodnota * i);};
    bool equals(DOLLAR *val){
      return (hodnota == val->hodnota);};
};

class FRANK : public MONEY {
  public:
    FRANK(int h){ hodnota = h;};
    FRANK *nasob(int i){
      return new FRANK (hodnota * i);};
    bool equals(FRANK *val){
      return (hodnota == val->hodnota);};
};
```

Duplicitný kód oboch metód `nasob` aj `equals` volá po odstránení.

Zovšeobecnený `equals`

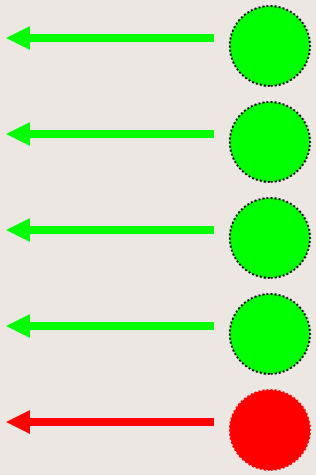
```
class MONEY {
  protected:
    int hodnota;
  public:
    bool equals(MONEY *val){
      return (hodnota == val->hodnota);};
};


class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){ hodnota = h;};
    DOLLAR *nasob(int i){
      return new DOLLAR(hodnota * i);};
};
class FRANK : public MONEY {
  public:
    FRANK(int h){ hodnota = h;};
    FRANK *nasob(int i){
      return new FRANK (hodnota * i);
    };
};
```

Metóda `equals` sa presunie do `MONEY`.

Jablká a hrušky

```
void testRovnostiDollarovFrankov(){
    FRANK patFrankov(5);
    DOLLAR patDolarov(5);
    assertTrue(patFrankov.equals(
        new FRANK(5)));
    assertFalse(patFrankov.equals(
        new FRANK(6)));
    assertTrue(patDolarov.equals(
        new DOLLAR(5)));
    assertFalse(patDolarov.equals(
        new DOLLAR(6)));
    assertFalse(patFrankov.equals(
        new DOLLAR(5)));
}
```



```
class MONEY {
  protected:
    int hodnota;
  public:
    bool equals(MONEY *val){
      return (hodnota == val->hodnota);};
};
```

Test nerovnosti medzi dolármi a frankami zlyhal.

Zavedieme meny

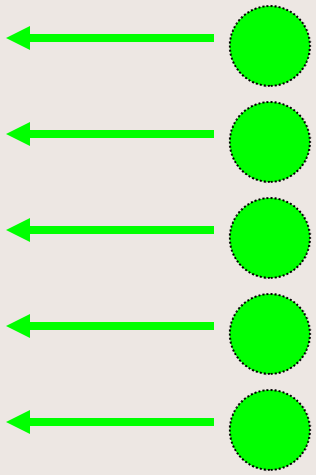
```
class MONEY {
  protected:
    int hodnota;
    string curr;
 public:
    bool equals(MONEY *val){
     if (curr != val->curr)
         return false;
     return (hodnota == val->hodnota);};
};
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){hodnota = h;
                  curr = "USD";};
    ...
};
class FRANK : public MONEY {
  public:
    FRANK(int h){ hodnota = h;;
                  curr = "CHF";};
   ...
};
```

Zavedieme novú premennú `curr` pre označenie meny.

Fake it!.

Jablká a hrušky ešte raz

```
void testRovnostiDollarovFrankov(){
    FRANK patFrankov(5);
    DOLLAR patDolarov(5);
    assertTrue(patFrankov.equals(
        new FRANK(5)));
    assertFalse(patFrankov.equals(
        new FRANK(6)));
    assertTrue(patDolarov.equals(
        new DOLLAR(5)));
    assertFalse(patDolarov.equals(
        new DOLLAR(6)));
    assertFalse(patFrankov.equals(
        new DOLLAR(5)));
}
```

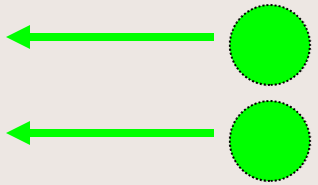


```
class MONEY {
 . . .
 public:
    bool equals(MONEY *val){
     if (curr != val->curr)
        return false;
     return (hodnota == val->hodnota);};
};
```

Sčítanie dolárov

```
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){hodnota = h; curr = "USD"; kurz=1;};
    MONEY *nasob(int i){
        return new DOLLAR(hodnota * i);};
    MONEY *scitaj(MONEY *val){
       return new DOLLAR(getVal() + val->getVal());}
};
```

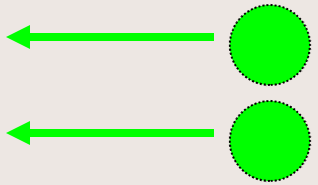
```
void testScitania(){
  DOLLAR patDolarov(5);
  DOLLAR desatDolarov(10);
  MONEY *vysledok = patDolarov.scitaj(new DOLLAR(5));
  assertTrue(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new DOLLAR(6));
  assertFalse(desatDolarov.equals(vysledok));
}
```



Sčítanie frankov

```
class FRANK : public MONEY {
  public:
 FRANK(int h){hodnota = h; curr = "CHF"; kurz=2;};
    MONEY *nasob(int i){
        return new FRANK(hodnota * i);};
    MONEY *scitaj(MONEY *val){
       return new FRANK(getVal() + val->getVal());}
};
```

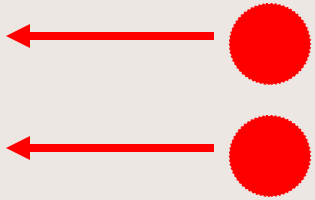
```
void testScitania(){
  FRANK patFrankov(5);
  FRANK desatFrankov(10);
  MONEY *vysledok = patFrankov.scitaj(new FRANK(5));
  assertTrue(desatFrankov.equals(vysledok));
  vysledok = patFrankov.scitaj(new FRANK(6));
  assertFalse(desatFrankov.equals(vysledok));
}
```



Sčítanie v rôznych menách

```
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){hodnota = h; curr = "USD"; kurz=1;};
    MONEY *nasob(int i){
        return new DOLLAR(hodnota * i);};
    MONEY *scitaj(MONEY *val){
       return new DOLLAR(getVal() + val->getVal());}
};
```

```
void testScitania(){
  FRANK patFrankov(5);
  DOLLAR patDolarov(5);
  DOLLAR desatDolarov(10);
  FRANK desatFrankov(10);
  MONEY *vysledok = patDolarov.scitaj(new FRANK(10));
  assertTrue(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new FRANK(5));
  assertFalse(desatDolarov.equals(vysledok));
}
```

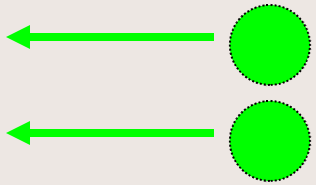


Zavedenie kurzu

```
class MONEY {
  protected:
    int hodnota;
    int kurz;
    string curr;
  public:
    bool equals(MONEY *val){
       if (curr != val->curr) return false;
       return (hodnota == val->hodnota);};
    int getVal(){return hodnota;};
    int getKurz(){return kurz;};
};
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){hodnota = h; curr = "USD"; kurz=1;};
    MONEY *nasob(int i){return new DOLLAR(hodnota * i);};
    MONEY *scitaj(MONEY *val){
       return new DOLLAR(getVal()/getKurz() + val->getVal()/val->getKurz());}
};
class FRANK : public MONEY {
  public:
    FRANK(int h){ hodnota = h;; curr = "CHF"; kurz=2;};
    MONEY *nasob(int i){return new FRANK (hodnota * i);};
    MONEY *scitaj(MONEY *val){
       return new DOLLAR(getVal()/getKurz() + val->getVal()/val->getKurz());}
};
```

Vylepšené sčítanie

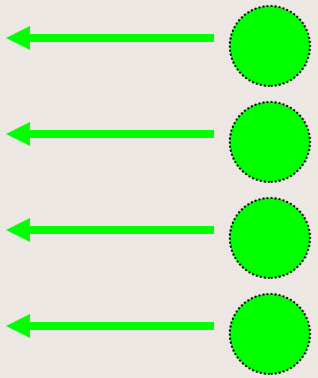
```
void testScitania(){
  FRANK patFrankov(5);
  DOLLAR patDolarov(5);
  DOLLAR desatDolarov(10);
  FRANK desatFrankov(10);
  MONEY *vysledok = patDolarov.scitaj(new FRANK(10));
  assertTrue(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new FRANK(5));
  assertFalse(desatDolarov.equals(vysledok));
}
```



```
class DOLLAR : public MONEY {
  public:
    DOLLAR(int h){hodnota = h; curr = "USD"; kurz=1;};
    MONEY *nasob(int i){
        return new DOLLAR(hodnota * i);};
    MONEY *scitaj(MONEY *val){
      return new DOLLAR(getVal()/getKurz()
                  + val->getVal()/val->getKurz());}
};
```

Všetky testy na sčítanie

```
void testScitania(){
  FRANK patFrankov(5);
  DOLLAR patDolarov(5);
  DOLLAR desatDolarov(10);
  FRANK desatFrankov(10);
  MONEY *vysledok = patDolarov.scitaj(new DOLLAR(5));
  assertTrue(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new DOLLAR(6));
  assertFalse(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new FRANK(10));
  assertTrue(desatDolarov.equals(vysledok));
  vysledok = patDolarov.scitaj(new FRANK(5));
  assertFalse(desatDolarov.equals(vysledok));
}
```



Nezávislosť

- **Testy musia byť od seba úplne nezávislé.**

- **Hodnoty premenných , ktoré by po skončení testu mohli použiť iné testy, musia byť uvedené do stavu, v ktorom boli pred začiatkom testu.**

- **Testy sa musia úplne ignorovať. Zlyhanie jedného testu nesmie ovplyvniť chod ďalších testov.**

- **Usporiadanie testov má byť ľubovoľné.**

Nové testy

- Nové testy napádajú pri programovaní neustále.
- **Píšte si zoznamy testov na kus papiera.** Realizované testy vyškrtávajte.
- **Tvorte si rôzne zoznamy testov:** aktuálne na tento deň či pre tento problém, testy pre blízku budúcnosť (týždeň), testy pred ukončením projektu, atď.

Pravidlo odkladania testov na neskoršie
Nikdy neodkladajte testy na nikdy.

Testy najprv

- **Testy píšte pred tým, ako programujete.**
- **Ak píšete testy najprv, redukujete stres z toho, či to bude fungovať.**

Testovacie dáta

- **Dáta pre testy si vyberajte jasné a zrozumiteľné. Až raz test zlyhá, každý by mal ľahko a hneď pochopiť, o čo išlo.**
- **Nepoužívajte v testoch tie isté konštanty.**
- **Používajte realistické dáta.**
- **Používajte dáta, ktoré sa budú používať pri skutočnom behu programu.**

Ako začať

Pred začiatkom budovania realistického testu musíme si ujasniť tri základné otázky:

- **Akú operáciu ideme testovať?**
- **Čo má byť korektný vstup?**
- **Čo má byť korektný výstup na tento vstup?**

```
void testRedukovanehoPolygonu{
   Reducer r = new Reducer(new Polygon());
   assertEquals(0, r.result().npoints);
}
```

Testovacie prostredie

- **Testy musia bežať rýchlo, aby ich časté spúšťanie nezdržovalo.**

- **Testy musia byť ľahko definovateľné a pridávateľné do zoznamu testov.**

- **Testy musia vypisovať jasné informácie, ktoré sú ľahko pochopiteľné a majú jasnú súvislosť so systémom aj príslušným testom.**

Svetlo na konci tunelu

Zelené pravidlo

Implementujte testy postupne, aby ste boli vždy na dosah zelenej.