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/

Prečo sa nám nechce refaktorovať?

1) Nie je nám jasné, ako refaktorovať.

Treba sa to naučiť. Najviac pomáha zdravý rozum.

2) Ak je prínos refaktorizácie dlhodobý, prečo sa namáhať teraz? Ktovie, či budem na programe vtedy ešte pracovať, aby som si plody vychutnal.

Refaktorovanie pripomína šport a zdravú výživu. Má rovnako krátkodobé ako dlhodobé účinky.

3) Refaktorovanie kódu je práca navyše. Platia nás za vyvíjanie nových vecí.

Refaktorovaním sa zlepšuje čitateľnosť a zrýchľuje tempo budúceho programovania.

4) Refaktorovaním sa môže existujúci program pokaziť.

To isté platí aj pri práci s neprehľadným, nelogickým a duplicitným kódom.

Katalóg refaktorizácií

Martin Fowler uvádza k knihe

Fowler, Martin, 2003: *Refaktoring (Zlepšení existujícího kódu)*, Grada Publishing, Praha

ako aj na webe na adrese http://www.refactoring.com/catalog

celý rad príkladov refaktorizácií, z ktorých som niektoré vybral:

Príklad objektu `null`

```
class Domacnost{
   Najomnik _najomnik;
  public:
   Najomnik getNajomnik(){
          return _najomnik;
   };
```

```
class Najomnik{
  public:
   string getMeno();
   PlanPlatenia getPlan();
   HistoriaPlatieb getHistoria();
   . . .
}
```

```
   . . .
   Najomnik najomnik = domacnost.getNajomnik();
   if (najomnik == null)
          menoNajomnika = "obyvatel";
    else
           menoNajomnika = najomnik.getMeno();
   if (najomnik == null)
          plan = PlanPlatenia.zakladny();
    else
          plan = najomnik.getPlan();
   . . .
}
```

V kóde musíme kontrolovať pri volaní každej metódy triedy `Zakaznik`, či je `null` a riešiť to náhradnými hodnotami.

Príklad objektu `null` – 2

```
class NajomnikNull: public Najomnik{
  public:
   bool isNull(){   return true; };
 string getMeno(){
      return "obyvatel";
   };
   . . .
}
```

```
class Najomnik{
 public:
   bool isNull(){ return false; };
. . .
}
class Domacnost{
   Najomnik _najomnik;
  public:
   Najomnik getNajomnik(){
          return (_najomnik == null) ? new NajomnikNull() : _najomnik;
   };
   . . .
}
```

```
   . . .
   Najomnik najomnik = domacnost.getNajomnik();
   menoNajomnika = najomnik.getMeno();
   . . .
}
```

Inicializácia namiesto priradenia

```
void foo() {
   int i;
   ....
   i = 7;
   ....
}
```

```
void foo() {
   ...
   int i = 7;
}
```

Inicializácia bez priradenia je zbytočná.

Hrozí použitie nenastavenej hodnoty.

C++ je pomalšie, pretože volá default konštruktor.

Odstráňte dvojitú negáciu

```
if ( !item.isNotFound() )
```

```
if ( item.isFound() )
```

Výmena poradia pri negatívnej podmienke

```
if ( !isSummer( date )){
  charge = winterCharge( quantity );
} else{
  charge = summerCharge( quantity );
}
```

```
if ( isSummer( date )){
  charge = summerCharge( quantity );
} else{
  charge = winterCharge( quantity );
}
```

Negatívna podmienka sa ťažšie číta a ľahšie prehliadne.

Redukujte priestor premennej

```
void foo(){
    int i = 7;

    // i sa nepoužíva

    if (...){
        // i sa používa
    }

    // i sa ďalej nepoužíva
}
```

Zbytočná možnosť použiť i na neplánovaný účel.

Nebezpečenstvo, že i ostane nenastavené.

```
void foo(){

    // i nemôže byť použité

    if (...){
        int i = 7;
        // i sa používa
    }

    // i nemôže byť použité
}
```

Náhrada rekurzie iteráciou

```
public void countDown(int n){
   if(n == 0) return;
   System.out.println(n + "...");
   waitASecond();
   countDown(n-1);
}
```

Rekurzia je elegantná ale v praxi sa zle predstavuje. Často je aj menej efektívna ako iterácia. Ak nie sú silné dôvody použiť ju, nepoužite ju.

```
public void countDown(int n){
   while(n > 0) {
         System.out.println(n + "...");
         waitASecond ();
         n -= 1;
   }
}
```

Rozdelenie cyklu

```
void printValues() {
   double averageAge = 0;
   double totalSalary = 0;
   for (int i = 0; i < people.length; i++) {
          averageAge += people[i].age;
          totalSalary += people[i].salary;
   }
   averageAge = averageAge / people.length;
   System.out.println(averageAge);
   System.out.println(totalSalary);
}
```

```
void printValues() {
   double totalSalary = 0;
   for (int i = 0; i < people.length; i++) {
          totalSalary += people[i].salary;
   }
   double averageAge = 0;
   for (int i = 0; i < people.length; i++) {
          averageAge += people[i].age;
   }
   averageAge = averageAge / people.length;
   System.out.println(averageAge);
   System.out.println(totalSalary);
}
```

Cyklus robiaci viac vecí zhoršuje čítanie.

Sťažuje ďalšiu refaktorizáciu.

Rozdelenie môže zrýchliť výpočet.

Rozdelenie cyklu 2.

```
void printValues() {
   System.out.println(averageAge());
   System.out.println(totalSalary());
}

private double averageAge() {
   double result = 0;
   for (int i = 0; i < people.length; i++) {
          result += people[i].age;
   }
   return result / people.length;
}

private double totalSalary() {
   double result = 0;
   for (int i = 0; i < people.length; i++) {
          result += people[i].salary;
   }
   return result;
}
```

Odstránenie dočasnej premennej

```
double zCena = zakazka.zakladnaCena();
return (zCena > 1000)
```

```
return (zakazka.zakladnaCena() > 1000)
```

Nahradenie dočasnej premennej volaním metódy

```
double zCena = mnozstvo * cena;
if (zCena > 1000){
        return zCena * 0.95;
}
else{
        return zCena * 0.98;
}
```

```
   if (zakladnaCena() > 1000){
        return zakladnaCena() * 0.95;
   }
   else{
        return zakladnaCena() * 0.98;
   }
. . .
int zakladnaCena(){
   return mnozstvo * cena;
}
```

Zavedenie vysvetľujúcich premenných

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
      (browser.toUpperCase().indexOf("IE") > -1) &&
       wasInitialized() && resize > 0 )
{
   // do something
}
```

```
boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
boolean isIEBrowser = browser.toUpperCase().indexOf("IE")> -1;
boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
   // do something
}
```

Zapuzdrenie premenných

```
int low, high;
boolean includes(int arg){
   return arg >= low && arg <= high;
}
```

Zapuzdrenie premenných Vám umožní sledovať, čo sa s premennou deje.

```
int low, high;

int getLow()  {return low;}
int getHigh() {return high;}

boolean includes (int arg) {
   return arg >= getLow() && arg <= getHigh();
}
```

Vloženie metódy do kódu

```
int getHodnotenie(){
     return (viacAkoPatRokovOmeskania()) ? 2 : 1;
}
int viacAkoPatRokovOmeskania(){
     return pocetRokov > 5;
}
```

```
int getHodnotenie(){
     return pocetRokov > 5 ? 2 : 1;
}
```

Odstránenie priradenia parametrom

```
int discount (int inputVal, int quantity, int yearToDate) {
   if (inputVal > 50) inputVal -= 2;
}
```

```
int discount (int inputVal, int quantity, int yearToDate) {
   int result = inputVal;
   if (inputVal > 50) result -= 2;
}
```

Privatizácia premennej

```
public:
        string _name;
```

```
  private:
        string _name;
  public:
        string getName() {return _name;}
        void setName(String arg) {_name = arg;}
```

Zákon zaslúženej pohromy

Kto zverejní číslo svojej kreditnej karty, zaslúži si byť okradnutý .

Náhrada magického čísla reťazcom

```
double potentialEnergy(double mass, double height){
   return mass * height * 9.81;
}
```

```
#define GRAVITATIONAL_CONSTANT 9.81                    /* C */

double potentialEnergy(double mass, double height) {
   return mass * GRAVITATIONAL_CONSTANT * height;
}
```

```
static double GRAVITATIONAL_CONSTANT = 9.81;           /* C++ */
double potentialEnergy(double mass, double height) {
   return mass * GRAVITATIONAL_CONSTANT * height;
}
```

```
public static double gravitationalConstant(){ return 9.81;}
                                                       /* Java */
double potentialEnergy(double mass, double height) {
   return mass * gravitationalConstant() * height;
}
```

Náhrada vnorených podmienok

```
double getPayAmount() {
   double result;
   if (_isDead) result = deadAmount();
   else {
         if (_isSeparated) result = separatedAmount();
         else {
            if (_isRetired) result = retiredAmount();
            else result = normalPayAmount();
         };
   }
   return result;
};
```

Zbavte sa nejasných vnorení.

```
double getPayAmount() {
   if (_isDead) return deadAmount();
   if (_isSeparated) return separatedAmount();
   if (_isRetired) return retiredAmount();
   return normalPayAmount();
};
```

Vyňatie spoločnej akcie

```
if (isSpecialDeal()) {
   total = price * 0.95;
   send();
}
else {
   total = price * 0.98;
   send();
}
```

```
if (isSpecialDeal())
   total = price * 0.95;
else
   total = price * 0.98;

send();
```

Spojenie rovnakej akcie

```
double disabilityAmount() {
   if (_seniority < 2) return 0;
   if (_monthsDisabled > 12) return 0;
   if (_isPartTime) return 0;
   . . .
```

```
double disabilityAmount() {
   if (isNotEligableForDisability()) return 0;
   . . .
```

Odstránenie príznaku

```
void bezpecnostnyTest(String[] people){
   bool najdeny = false;
   for (int i = 0; i < people.length; i++){
      if (!najdeny){
          if (people[i].equals ("Don")){
                    poslatVarovanie();
                    najdeny = true;
          }
          if (people[i].equals ("John")){
                    poslatVarovanie();
                    najdeny = true;
          }
      }
   }
}
```

```
void bezpecnostnyTest(String[] people){
   for (int i = 0; i < people.length; i++){
          if (people[i].equals ("Don")){
                    poslatVarovanie();
                    break;
          }
          if (people[i].equals ("John")){
                    poslatVarovanie();
                    break;
          }
      }
   }
}
```

Nahradenie chybového kódu výnimkou

```
int withdraw(int amount) {
   if (amount > _balance) {
         return -1;
   }
   else {
         balance -= amount;
         return 0;
   }
}
```

Chybová hodnota nezaručuje zachytenie chyby

```
void withdraw(int amount) throws BalanceException {
   if (amount > _balance) {
         throw new BalanceException();
   }
   balance -= amount;
}
```

Nahradenie falošnej výnimky testom

```
double getValueForPeriod (int periodNumber) {
   try {
         return _values[periodNumber];
   } catch (ArrayIndexOutOfBoundsException e) {
         return 0;
   }
}
```

Využívajte výnimky na výnimočné, neočakávané situácie. Nemali by sa používať ako náhrada testovacích podmienok.

```
double getValueForPeriod (int periodNumber) {
   if (periodNumber < 0 || periodNumber >= _values.length){
         return 0;
   }
   return _values[periodNumber];
}
```

Parametrizuj metódu

```
void patPercentZvysenie(){
        plat *= 1.05;
}
void desatPercentZvysenie() {
        plat *= 1.1;
}
```

```
void zvyseniePlatu(percenta) (){
        plat *= percenta;
}
```

Odstránenie metódy pre zápis

```
class Ucet{
        string _id;
  public:
        Ucet(string id){
                setId(id);
        };
        void setId(string id){
                _id = id;
        }
        . . .
};
```

```
class Ucet{
        string _id;
  public:
        Ucet(string id){
                _id = id;
        };
        . . .
};
```

Oddelenie modifikácie od čítania

```
string najdiOsobu(String[] people){
   for (int i = 0; i < people.length; i++){
          if (people[i].equals ("Don")){
                    ulozOsobu("Don");
                    return "Don";
          }
          if (people[i].equals ("John")){
                    ulozOsobu("John");
                    return "John";
          }
      }
   }
}
```

```
void najdiOsobu(String[] people){
   for (int i = 0; i < people.length; i++){
          if (people[i].equals ("Don")){
                    ulozOsobu("Don");
          }
          if (people[i].equals ("John")){
                    ulozOsobu("John");
          }
      }
   }
}
string vratNajdenuOsobu(){
          return ulozenaOsoba();
}
```

Poskytnutie potrebných informácií

```
void printValues() {
   for (int i = 0; i < people.length; i++) {
       System.out.println(people[i].name+" has salary"+people[i].salary);
   }
}
public static void main(String args[]) {
          ...
   printValues();
}
```

Parametrizácia zlepšuje čitateľnosť a umožňuje použitie metódy aj v odlišnom kontexte.

```
void printValues(PrintStream outfile) {
   for (int i = 0; i < people.length; i++) {
       outfile.println(people[i].name+" has salary "+people[i].salary);
   }
}
public static void main(String args[]) {
          ...
   printValues(System.out);
}
```

Nahradenie algoritmu

```
String foundPerson(String[] people){
   for (int i = 0; i < people.length; i++) {
        if (people[i].equals ("Don")){
              return "Don";
        }
        if (people[i].equals ("John")){
              return "John";
        }
        return "";
}
```

```
String foundPerson(String[] people){
   List candidates = Arrays.asList(new String[]{"Don","John"});
   for (int i=0; i<people.length; i++)
        if (candidates.contains(people[i])){
              return people[i];
        }
   return "";
}
```

Zjednodušenie volania metód

Je viacero spôsobov, ako zjednodušiť volanie metód:

- zrozumiteľnejšie premenovanie metódy,
- pridávanie a uberanie parametrov,
- oddelenie modifikácie od čítania,
- zlučovanie metód pomocou parametrizácie,
- nahrádzanie parametru metódou,
- zavádzanie objektu pre parametre,
- odstránenie prístupovej metódy pre zápis (iba cez konštruktor),
  ukrývanie metódy (cez `private`),
- zapúzdrenie pretypovania na potomka,
- nahradenie chybového kódu výnimkou,
- nahradenie falošnej výnimky testom,

Postup pri zmene volania metódy

1) Zistite, či deklaráciu metódy neimplementuje aj niektorý predok alebo potomok menenej triedy.

2) Deklarujte novú metódu s novým menom. Skopírujte telo pôvodnej metódy do tela novej metódy a preveďte ostatné zmeny.

3) Zmeňte telo pôvodnej metódy tak, aby volala novú metódu.

4) Nájdite všetky volania pôvodnej metódy a zameňte ich volaním novej metódy.

5) Odstráňte pôvodnú metódu.

6) Ak je pôvodná metóda súčasťou rozhrania a nemôžete ju odstrániť, ponechajte ju a označte ako zastaralú (`deprecated`)