Unit testy a refactoring /
JUnit

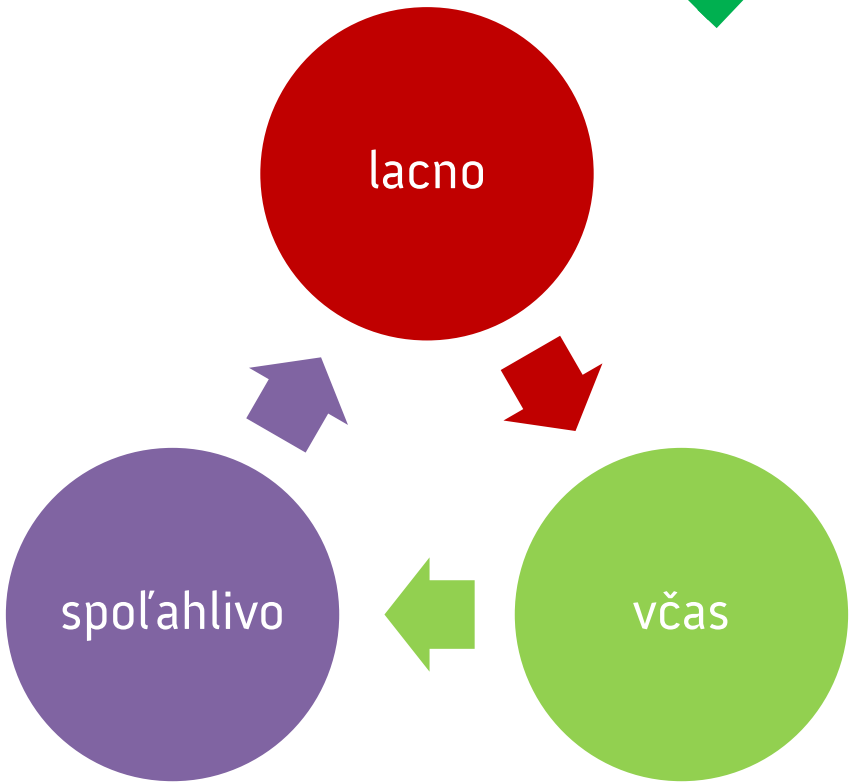
SPR1a

Róbert Novotný
robert.novotny@upjs.sk

28. 2. 2011

Problémy softvérového vývoja

- softvér je drahý
- nespoľahlivý
- často sa nenasadí
  - len 2% dodaného softvéru sú použiteľné bez zmien!
  - 47% softvéru bolo doručeného, ale nepoužitého!



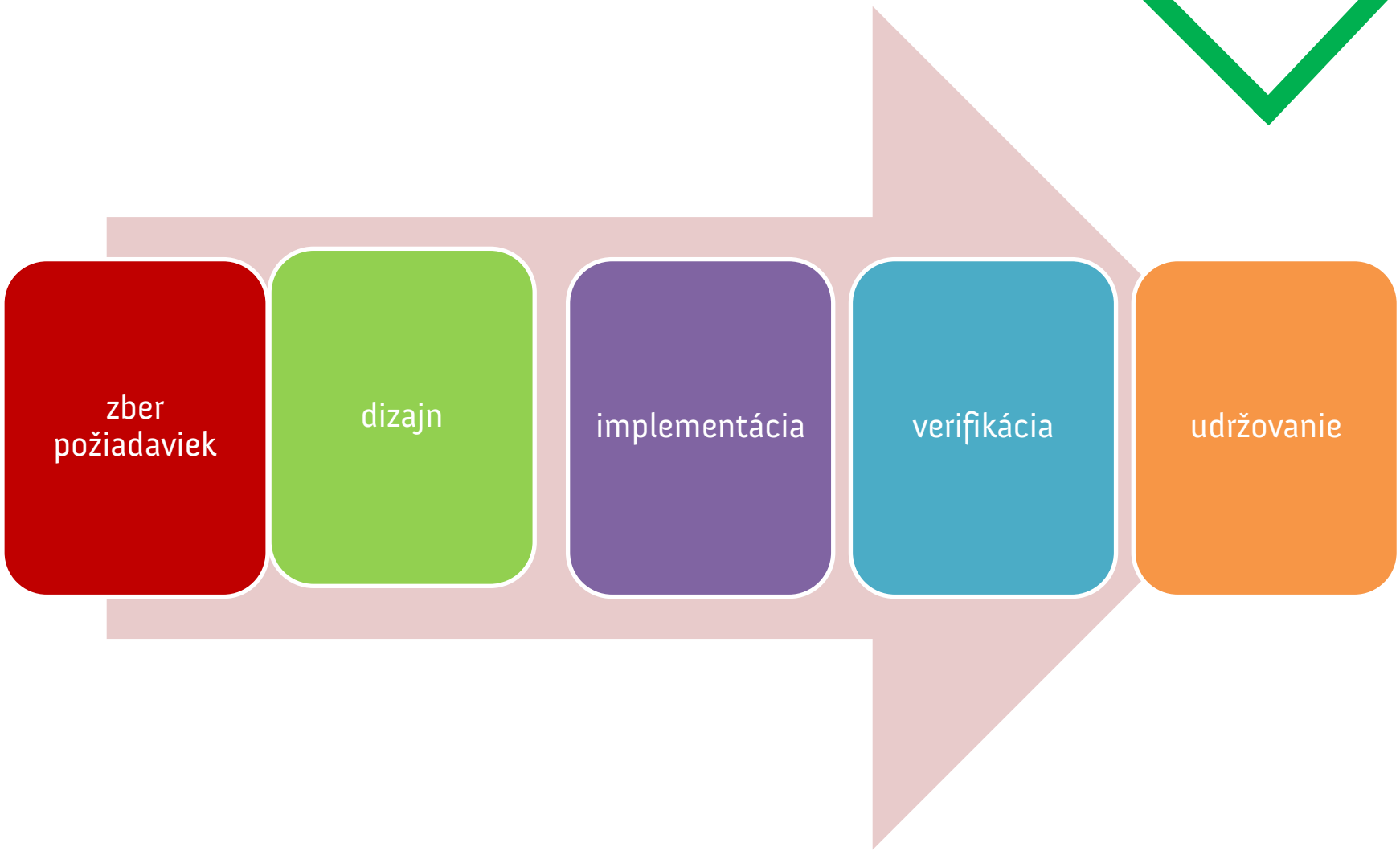
Softvér je sčasti umenie

- **agilné techniky** = dramatická zmena myslenia v softvérovom vývoji oproti zaužívaným technikám
- manifest agilného vývoja softvéru
  - cesty lepšieho vývoja softvéru sa odhalia len vývojom a napomáhaním ostatným pri vývoji

Manifest agilného vývoja

- **jedinci a interakcie** nad procesmi a nástrojmi
- **fungujúci softvér** nad vyčerpávajúcou dokumentáciou
- **spolupráca so zákazníkom** nad vyjednaním kontraktu
- **reakcie na zmeny** nad nasledovaním plánu

Klasický vývoj: waterfall



Klasický problém: čo ak sa zmenia požiadavky uprostred vývoja

- najľahšie riešenie: lepiaca páska
- ak aj nie, tak každý softvér časom „hnije" (software rot)

Obava zo zmeny

- základná zásada: čo funguje, do toho nebabrem
- obava robiť veľké dizajnérske zásahy
- ak je čas na väčšiu zmenu, radšej sa uprednostní lepenie
  - duplikovanie kódu
  - zmeny typu „len nech to funguje"

Zásady extrémneho programovania pre kódenie

- **Extreme Programming** (1996): jedna z radikálnejších agilných techník
  - extremeprogramming.org
- zásady:
  - každá trieda musí mať test = unit test
  - najprv píšeme unit test, potom samotný kód

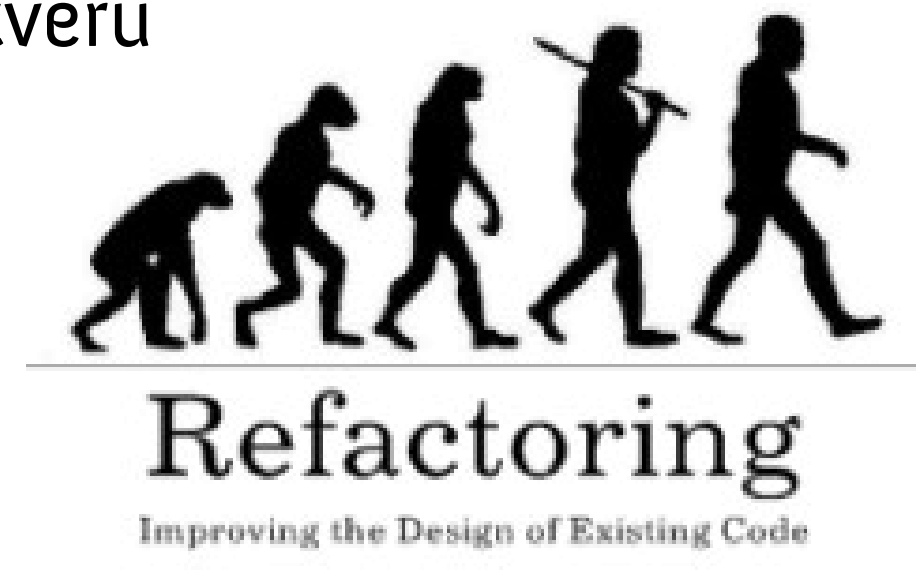
Typický unit test

- unit test je trieda testujúca triedu
- pre každú (verejnú) metódu porovnám
  - jej návratovú hodnotu
  - s očakávanou hodnotou
- test prejde, ak sa obe hodnoty zhodujú

Refactoring: alias načo sú dobré unit testy?

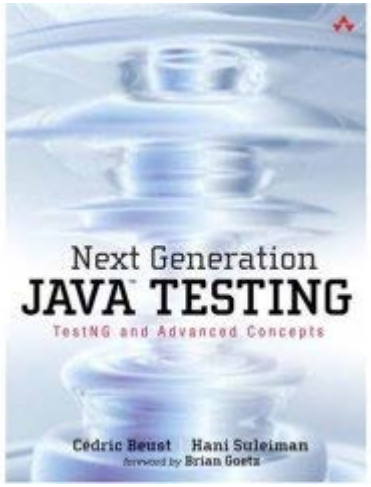
- unit test **zaručuje**, že trieda funguje korektne
- môžem meniť vnútro a nebojím sa, že „to pobabrem"
- možno teda upravovať jej vnútro a predchádzať hnitiu softvéru

refactoring = vylepšovanie kódu so zachovaním funkcionality



Unit testing frameworky

- uľahčujú písanie unit testov
- zelegantňujú ich hromadné spúšťanie
- Java:
  - JUnit, ...
  - TestNG, ...



DEMO

Refactoring: alias načo sú dobré unit testy?

- unit testy sa dajú spúšťať **automaticky**
  - pred vypustením verzie
- podporuje sa priebežná integrácia (**continuous integration**)
  - každý deň sa zbuilduje, otestuje a nasadí celý projekt
  - pokazené unit testy sa dajú ihneď opraviť
  - problémy sa odhalia ihneď a nie až pri odovzdaní zákazníkovi

Riešenie chýb: postup prác

1. používateľ odhalí a zareportuje chybu...
2. ... chyba sa opraví...
3. ... napíše sa unit-test!

- predchádza sa **regresiám**
  - chyba sa opraví vo verzii X a znovuobjaví sa v neskoršej verzii

Načo písať testy?

- „code test first" uľahčuje návrh API
  - najprv povieme, čo od triedy chceme a potom povieme ako sa to má spraviť
  - máme lepší pohľad klienta triedy
- do triedy dodávame len veci, ktoré sú bezpodmienečne nutné
  - žiadny overengineering
- test zároveň zrkadlí požiadavky

Načo písať testy?

NAMIESTO ZABÍJANIA ČASU TESTAMI SI MI MAL UŽ DÁVNO DODAŤ FUNKCIONALITU!

Svet, kde nejestvujú testy

- ignorovanie testov získa čas na písanie funkcionality
- tým pádom ale nemôžeme refaktorovať
  - nemáme záruku, že niečo nepobabreme
- začína software rot
- v krátkodobom horizonte sme získali čas, ale v dlhodobom ho stratíme

Svet, kde nejestvujú testy

Zásady

- testujeme obvykle len public metódy
  - testujeme kontrakt, teda **čo** trieda robí, nie **ako** to robí
- netestujeme triviality a zbytočnosti
  - gettre, settre v Jave
- nespoľahlivé a nedeterministické objekty nahrádzame **mock** objektami (atrapami)