UINF/PAZ1c
Programovanie, algoritmy, zložitosť

motívy predmetu

- pokročilé OOP
- dizajn a udržovateľnosť programov
- príklady z praxe
- Java
- okienkové aplikácie (JavaFX)
- databázy
- Webové aplikácie (Angular + REST server)

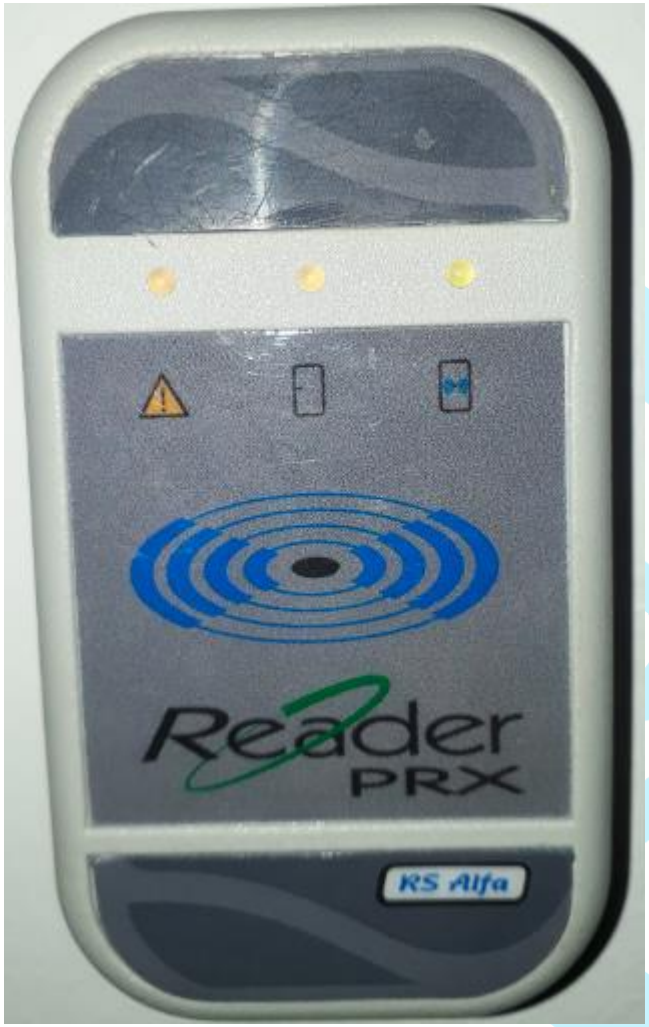
požiadavky na hodnotenie

- http://paz1c.ics.upjs.sk/
- plusové body za projekt
- mínusové body za
  - viac ako 3 neúčasti na cvikách
  - plagiátorstvo

projekt

- dvojčlenné tímy
- fázy:
  - 17. október: špecifikácia projektu zaslaná mailom
  - 2.11. : predstavenie DB návrhu projektu
  - 21.12. : prezentácia I. + pridelenie spriateleného projektu
  - začiatok februára: prezentácia II.
    - dorobenie funkcionality do spriateleného projektu
    - možnosť opravy chýb z prvej prezentácie za polovičné body

Dizajn a implementácia tried za pomoci unit testov


Reader
PRX
RS Alfa


STUDENT

projekt:
Prístupový systém

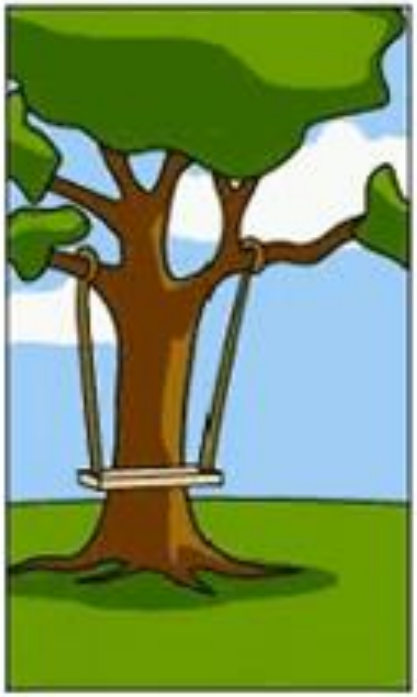
pozbierajme používateľské požiadavky

- po prečítaní čipu zapípa
- keď priložím správny čip, otvorí dvere
- čítačka musí byť čierna
- admin pridáva nové čipy do prístupového systému
- chcem vedieť, kto má ktorý čip
- ak niekto stratí čip alebo odíde z firmy/školy/bytovky, chcem ho vedieť zo systému odstrániť
- chcem mať záznamy o tom, kto kedy otvoril dvere
- ak vypadne elektrina, systém musí fungovať

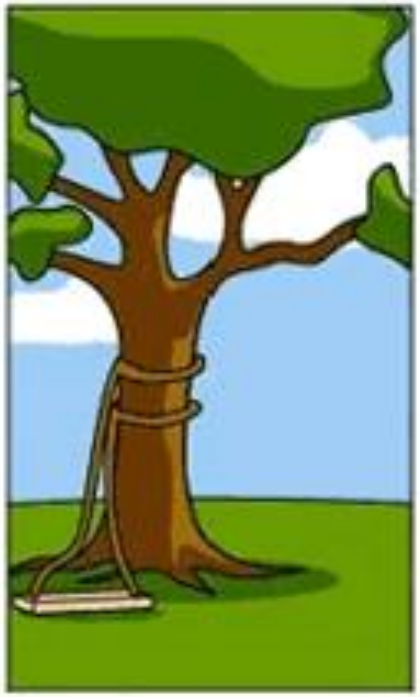
softvérová analytička formalizuje používateľské požiadavky

čo zákazník vysvetlil

čo naozaj chcel



čo pochopil analytik

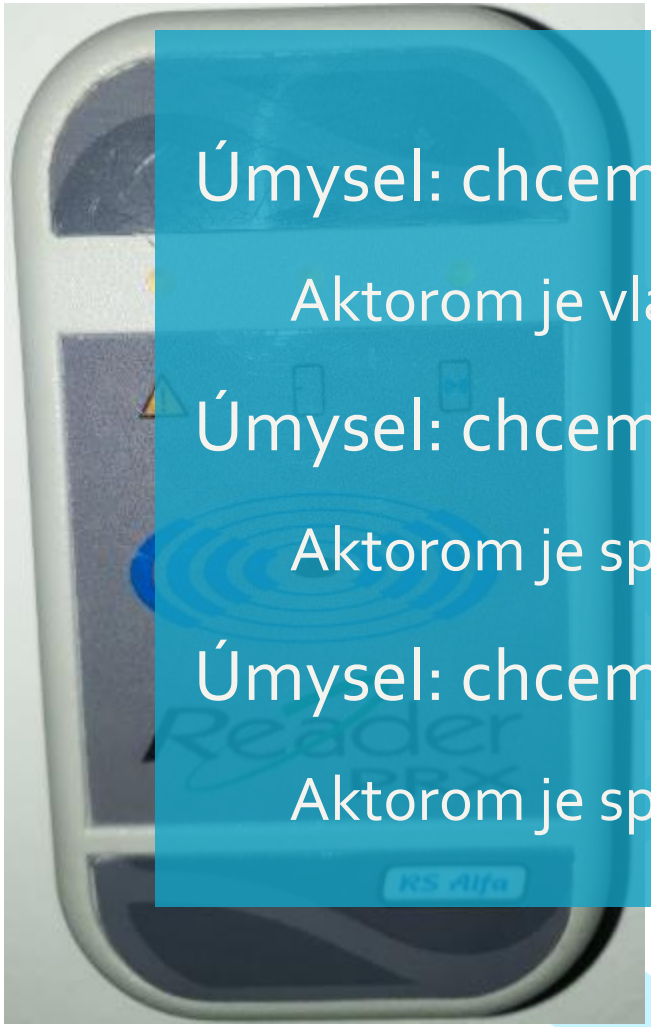


čo bolo nakódené

Use-case

zoznam krokov, ktoré vykonáva
aktor so systémom
v úmysle dosiahnuť svoj cieľ.

ivar jacobson



Úmysel: chcem otvoriť dvere

- Aktorom je vlastník čipu

Úmysel: chcem pridať čip

- Aktorom je správca systému

Úmysel: chcem vidieť príchody

- Aktorom je správca systému...

use-case: chcem pridať čip

1. Priložím čip k čítačke
2. Zadám údaje o vlastníkovi
3. Stlačím tlačidlo uloženia

use-case: chcem pridať čip

1. Priložím čip k čítačke
2. Zadám údaje o vlastníkovi
3. Stlačím tlačidlo uloženia

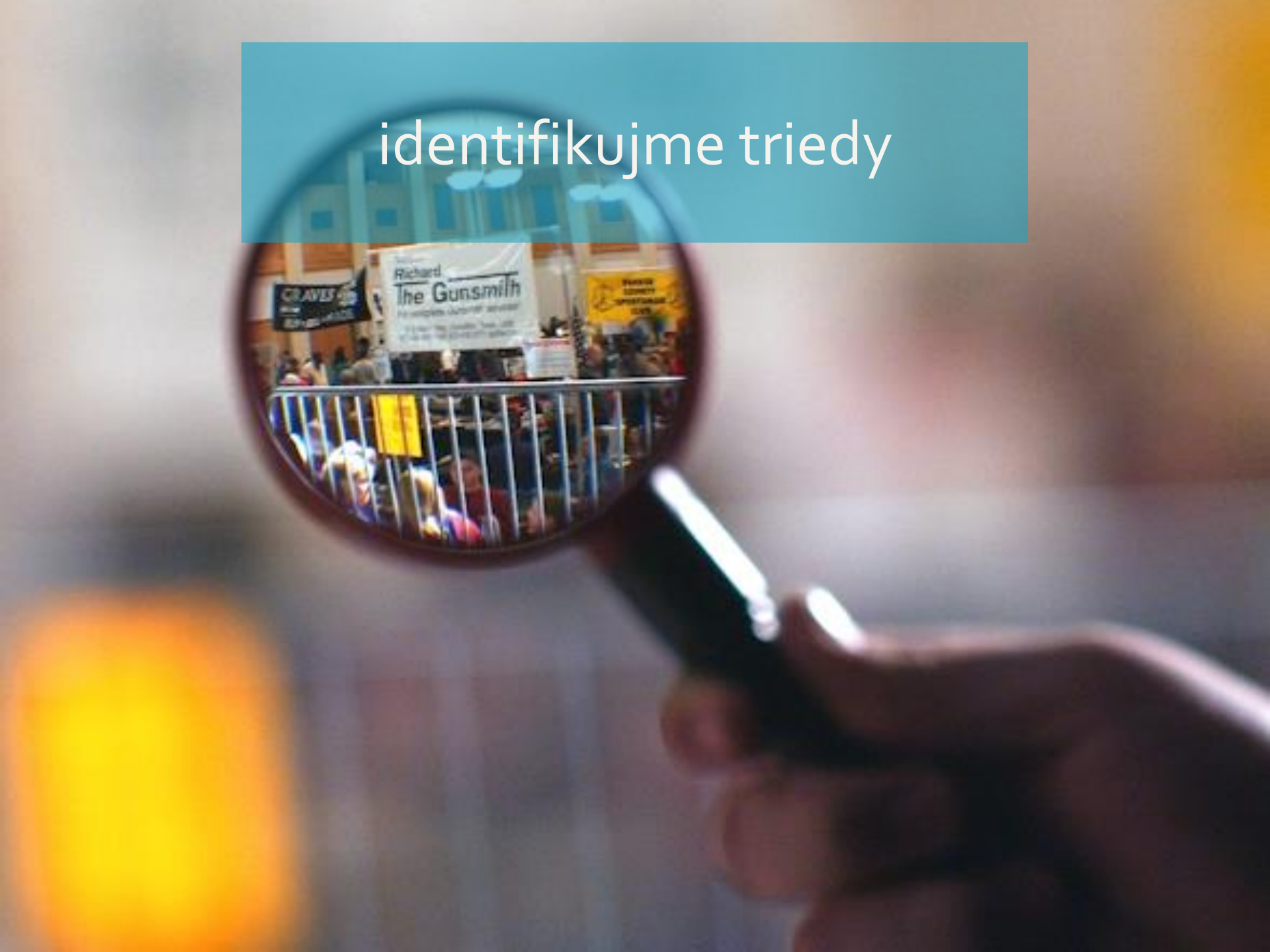
Optimistický scenár !

Je možné donekonečna vylepšovať a pokrývať záchranné, alebo neúspešné scenáre

use-case: chcem pridať čip

1. Priložím čip k čítačke
   1.1. Čip sa nepodarilo načítať – 2x zapípaj
   1.2. Ak čip už je v systéme, zobraz informácie o vlastníkovi
2. Zadám údaje o vlastníkovi
3. Stlačím tlačidlo uloženia
   2.1. ak údaje nemajú požadované vlastnosti, vráť sa na krok 2

identifikujme triedy



identifikujme triedy

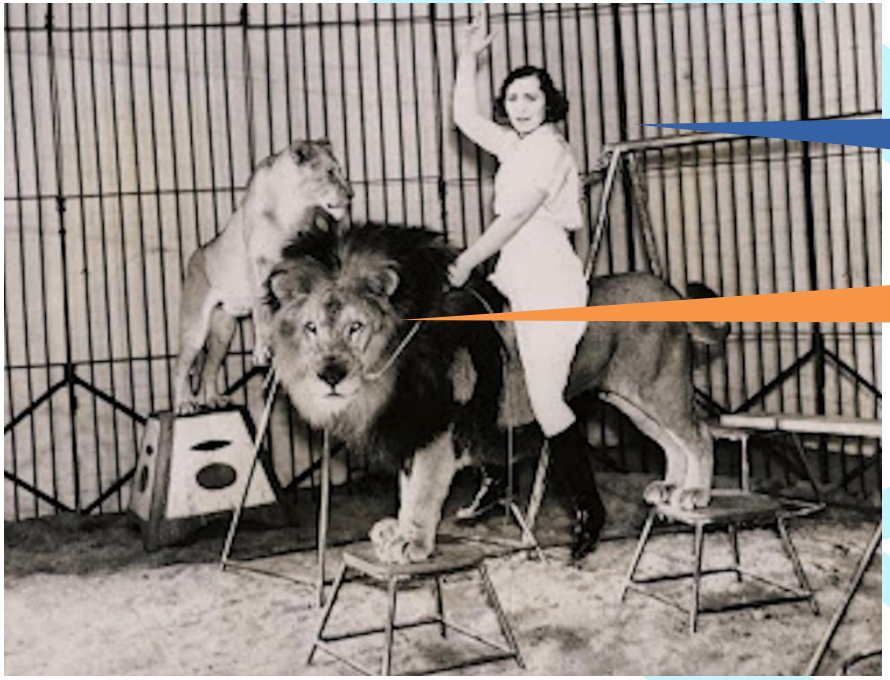
podstatné mená sú kandidáti pre triedy

- chcem prikladať čip k čítačke
- po priložení zaznie pípanie
- zadávam údaje o vlastníkovi
- admin pridáva nové čipy do prístupového systému
- stlačím tlačidlo uloženia
- chcem mať záznamy o tom, kto kedy otvoril dvere

Trieda by mala mať schopnosti a stav!

najprv schopnosti!

schopnosť = rozkaz = metóda



skáč!

zlez!

potom stav

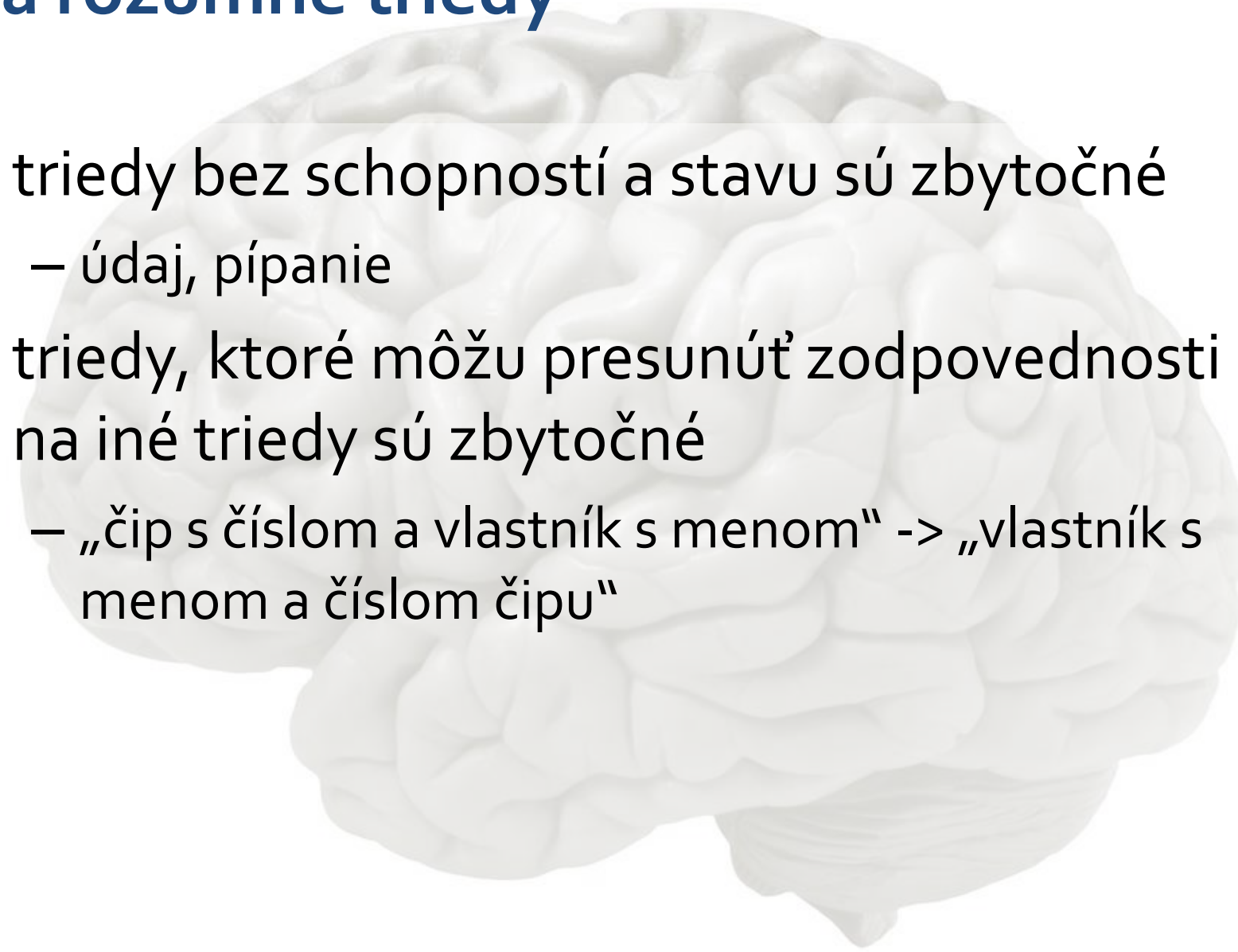
stav = inštančná premenná

druh: lev

farba: #ADADAD

meno: Dunčo

iba rozumné triedy



- triedy bez schopností a stavu sú zbytočné
  - údaj, pípanie
- triedy, ktoré môžu presunúť zodpovednosti na iné triedy sú zbytočné
  - „čip s číslom a vlastník s menom“ -> „vlastník s menom a číslom čipu“

dobrý návrh nad zlato

- odhalenie tried je základom dobrého návrhu
- umenie toho, čo vybrať a čo zahodiť
  - **skúsenosti** využívajú *best practices* z minulých projektov
  - **talent** predchádzať chybám
  - **šťastie**, vďaka ktorému projekt odolá zmenám


uniPaas
And So
You Code

dôraz na udržovateľnosť

- s narastajúcou veľkosťou projektu sa zvyšujú nároky na **udržovateľnosť**
  - udržovateľný (objektový) návrh je nutnosť
- programátor vyvíja softvér **pre ostatných**, nie pre seba
  - snaha predísť write-only kódu
- väčší projekt je skladačka, ktorá funguje, iba **ak jej časti robia to, čo robiť majú**
  - ako vieme, čo funguje a čo nie?

Ako otestovať našu triedu?

- Java začiatočník: vytvorí metódu **main()**
- výstupy riešime cez System.out.println()
- kontrolu urobíme od oka
- ak vidíme výstup, je to OK

Lesk a bieda main()

- Čo ak máme viac variantov použitia?
- V triede môže byť len jedna metóda main()!
- **Hlúpe riešenie** č. 1:
  - Podľa potreby odkomentovávame a zakomentovávame výseky kódu.
- **Hlúpe riešenie** č. 2:
  - vytvoriť viac testovacích tried
  - každá má svoj **main()**
  - spúšťame podľa potreby

Meditácie nad main()om

- Čo ak potrebujeme testovať **viacero vstupov**?
  - Budeme čítať z klávesnice dokola?
    - Kedy nás to prestane baviť?
- Kedy nás prestane baviť **okom kontrolovať** výstupy vo veľkom softvéri?
- Prečo má byť **testovací kód na kope** s reálnym kódom?
  - V prípade, že máme **main()** rovno v testovanej triede...

Chyby, chyby, chyby

- každý softvér obsahuje chyby!
  - *Code Complete*: 15-50 chýb na 1000 riadkov dodaného kódu
- testovaním predchádzame chybám
- viacero druhov testov a viacero klasifikácií

~~Kto neprogramuje, nech ani neje!~~
Kto netestuje, nech ani neprogramuje!

Rozličné druhy testov

Účel

- korektnosť
- výkonnosť
- spoľahlivosť
- zabezpečenie

Životný cyklus

- testovanie fázy požiadaviek
- testovanie fázy dizajnu
- testovanie fázy programovania
- vyhodnocovanie výsledkov testov
- testovanie fázy inštalácie
- akceptačné testy
- testy fázy udržovania

Zámer

- jednotkové testy
- testovania komponentov
- integračné testy
- systémové testy

Unit Testy!

Unit testy / Jednotkové testy

Robí kód naozaj to, čo chceme aby robil?

- testovanie jednotiek kódu na korektnosť použitia
- **unit** = najmenšia testovateľná časť aplikácie
- v OOP = **test triedy** / rozhrania
  - v procedurálnom programovaní: test procedúry
- testujeme malé, izolované časti funkcionality
- ak fungujú malé časti, budú fungovať aj tie väčšie

Základná idea unit testov

- pre testované metódy definujeme
  - testovací vstup
  - očakávaný výstup
- **test:** je **vypočítaný** výstup zhodný s **očakávaným** výstupom?
  - ak áno, test uspel
- definujeme toľko testovacích vstupov a očakávaných hodnôt, koľko treba.

Unit testy a JUnit

- JUnit – de facto štandardný framework pre unit testy v Jave
- zabudovaná podpora v každom schopnom IDE: Eclipse / IntelliJ / NetBeans

org.junit

Aktuálne Junit 5 (jupiter)

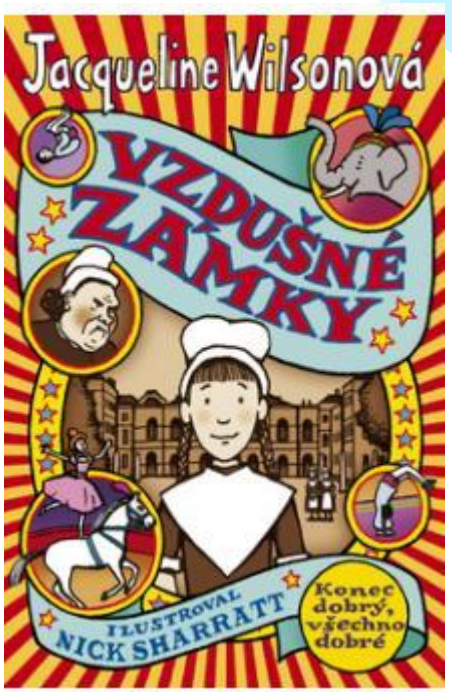
- milión metód pre porovnanie očakávaného a vypočítaného vstupu

assertEquals() assertNull() assertNotNull()
assertFalse() assertTrue() fail()

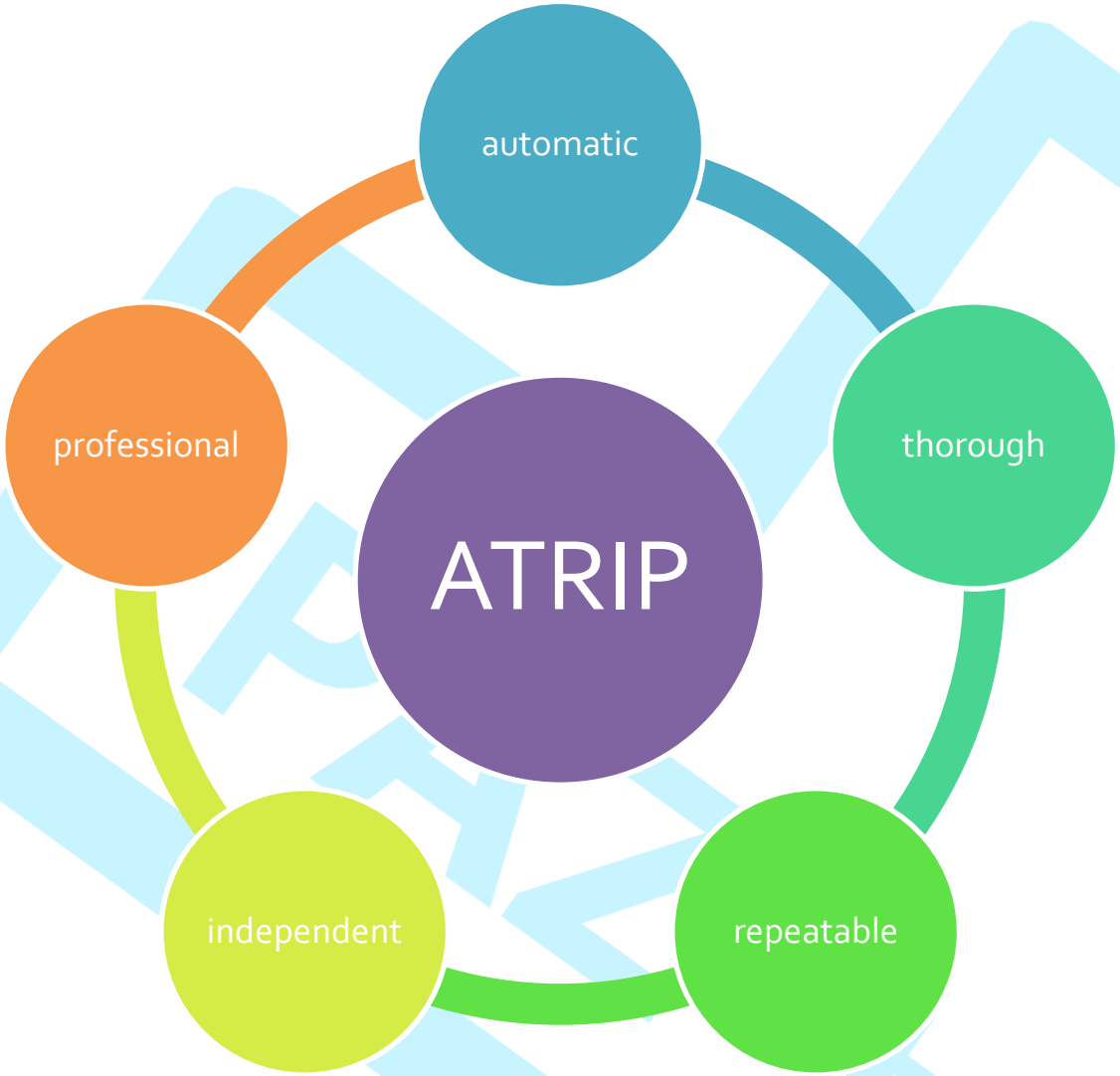
assertThrows(IllegalArgumentException.class, f())

Poznámky k unit testom

- každá verejná metóda triedy má mať unit test
- v **ideálnom stave** otestujeme správanie pre:
  - každý riadok kódu
  - každú vetvu kódu
  - každú výnimku, ktorá môže nastať
- obvyklá **realita**:
  - typické hlúpe a hraničné vstupy
  - chýbajúce či nesprávne dáta
  - ...



Vlastnosti dobrého testu - ATRIP



Vlastnosti dobrého testu - ATRIP

- automatic(ký)
  - máme mať možnosť spustiť ich pred vydaním verzie automaticky
- thorough (podrobný)
  - cieľ: 100% code coverage
  - pokrytie kódu
- repeatable (opakovateľný)
  - pri každom behu rovnaké výsledky
  - nezávislý od vonkajšieho prostredia

Vlastnosti dobrého testu - ATRIP

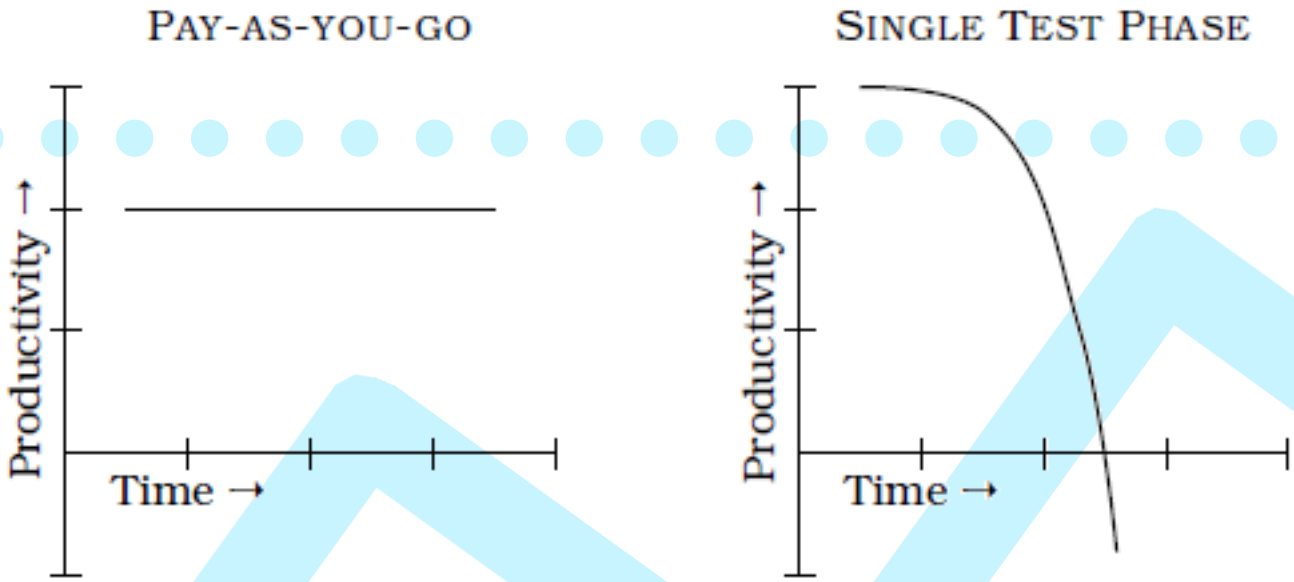
- independent (nezávislý)
  - testy sú navzájom nezávislé
  - v jednom teste overujeme len jednu konkrétnu vlastnosť
- professional
  - unit testy sú rovnako dôležité ako zvyšok kódu
    - pomer riadkov kódu k riadkom testu: 1:1, možno aj viac
  - netestujeme zbytočnosti
    - testy jednoriadkových getterov, setterov, prázdnych konštruktorov

Testovanie zabíja čas!

- ~~čas, ktorý zabijem písaním testu môžem venovať ďalším vlastnostiam!~~

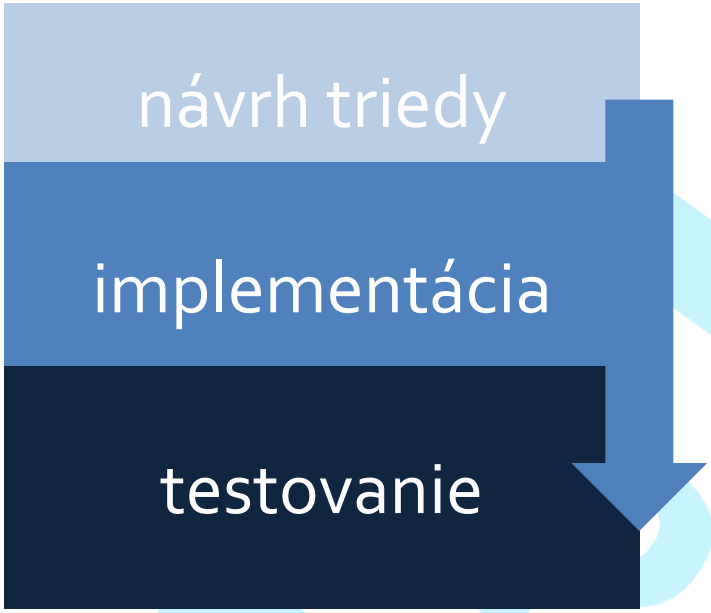
Mýtus od roku 20xx!

- pretože čas investovaný do testov sa vráti v dlhodobom horizonte

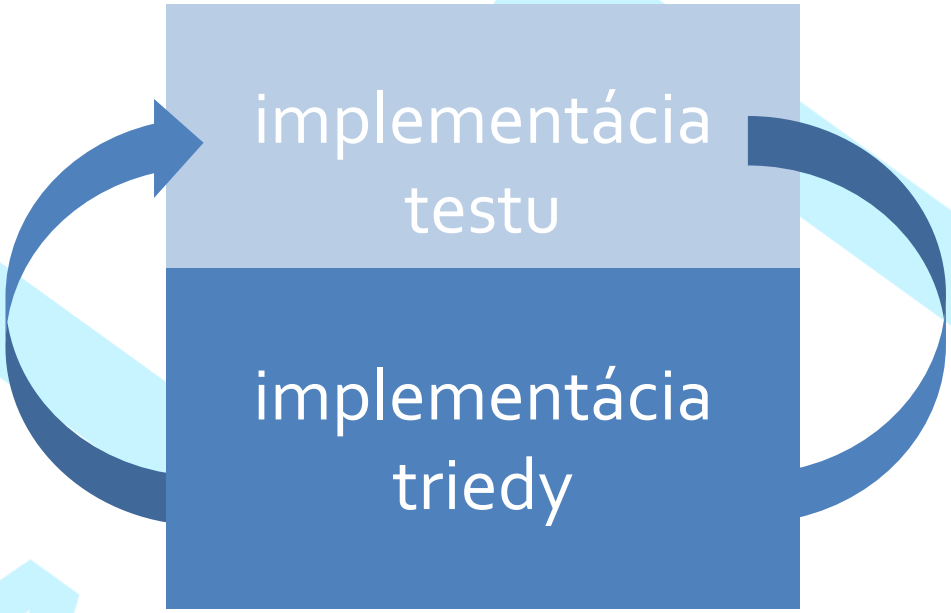


- klasický spôsob: testujeme na záver projektu
- pri použití testov:
  - vyššia priebežná investícia
  - tá je však konštantná
  - ale menej priebežných chýb: vždy pracujeme s otestovaným kódom!

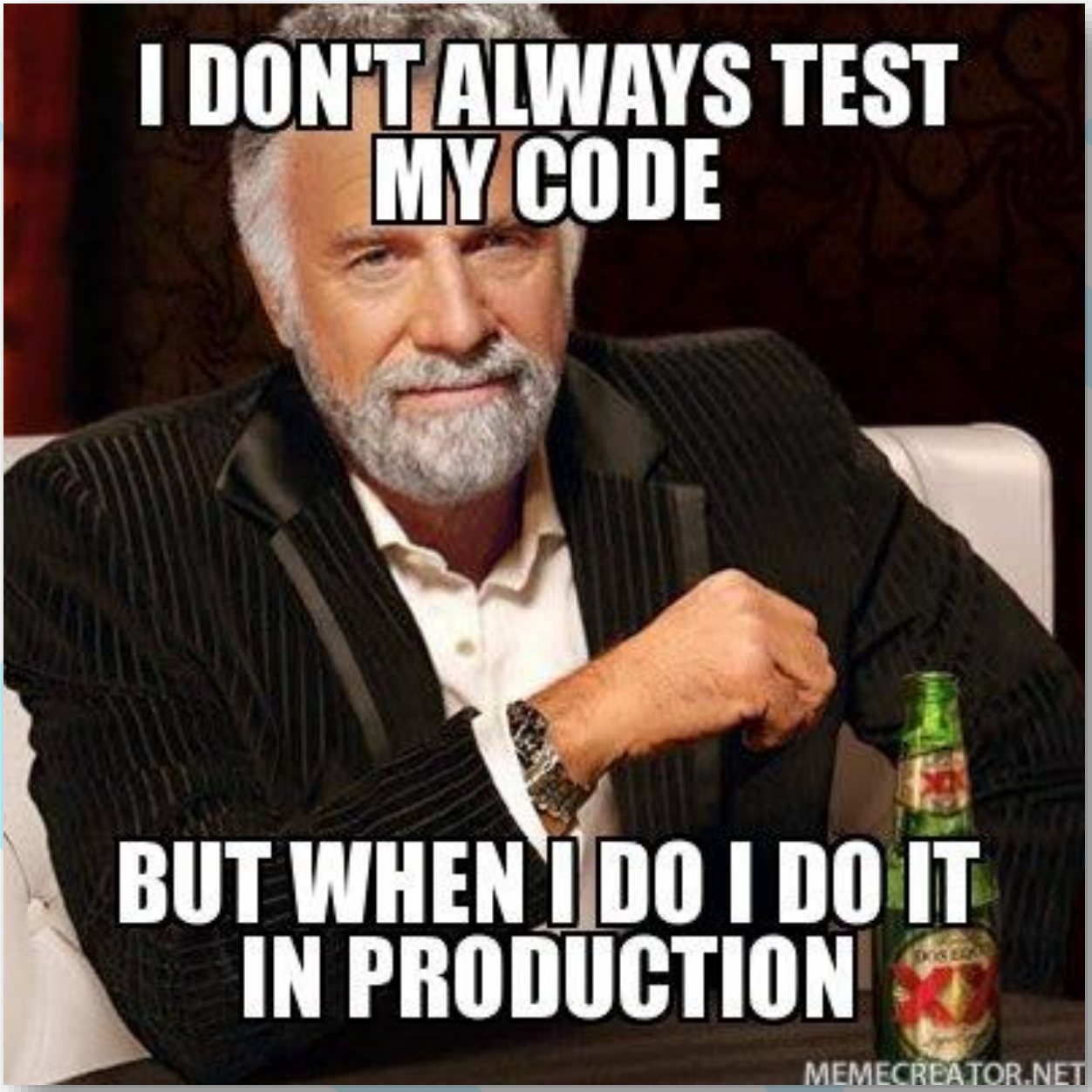
Napíšte najprv testy!

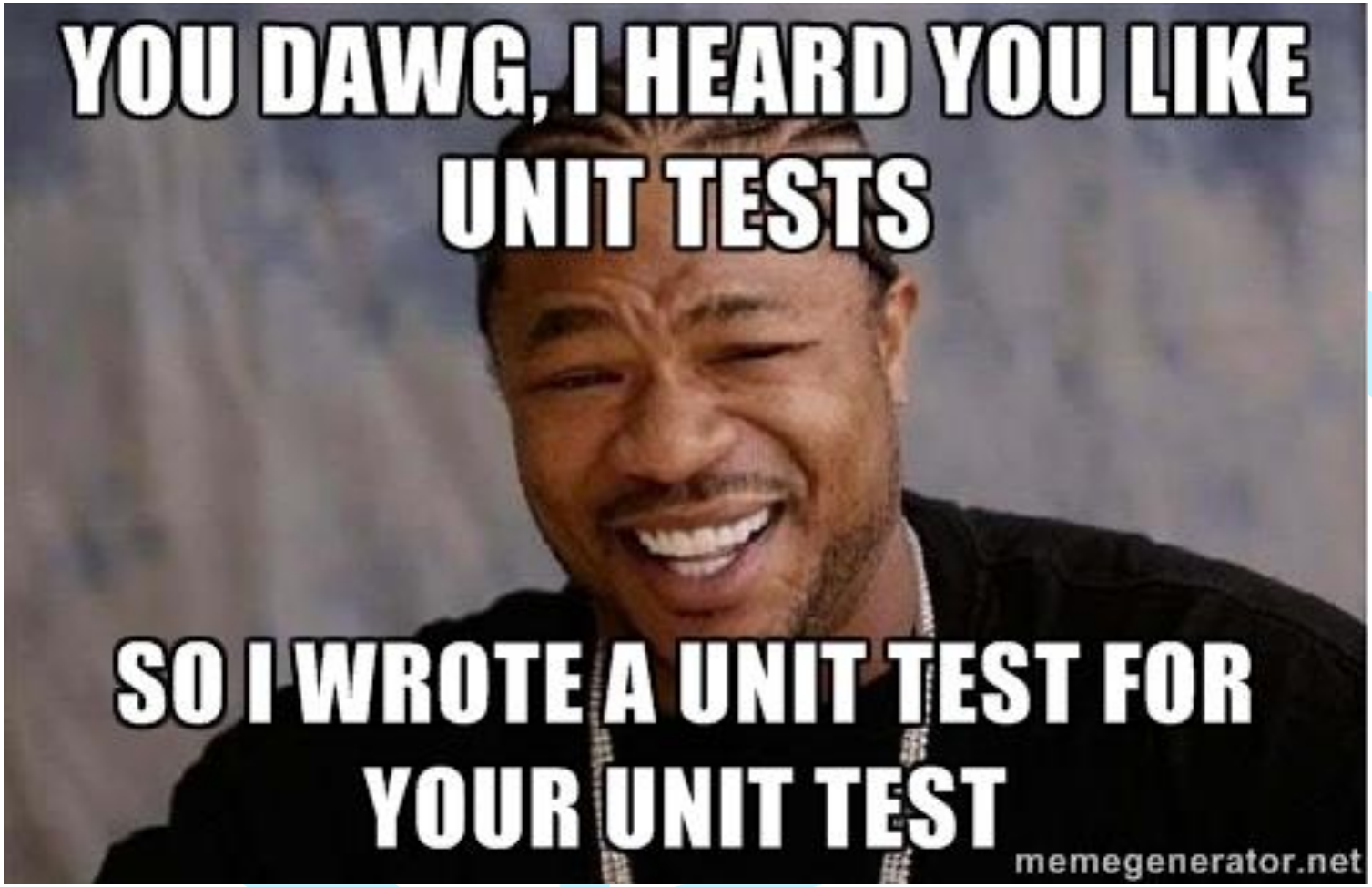


klasický postup



agilný postup


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
MEMECREATOR.NET


YOU DAWG, I HEARD YOU LIKE UNIT TESTS
SO I WROTE A UNIT TEST FOR YOUR UNIT TEST
memegenerator.net

odnesme si domov

- zozbierajme prípady použitia
- identifikujme triedy podstatnými menami
- identifikujme
  - 1. správanie
  - 2. stav metód
- najprv píšme test, potom kód

Otázky?

Otázky?

Žiadne.