

závislosti ActionListener
interface problém inštanciu
icon schopnosti komponentov
môže Automat výnimky aplikácie
add(new Kružnica triedy extends static vieme
Riešenie tried dáta OOP CitatDao prvkov
meniť metóda biznis
máme kóda DAO pre public test JFrame
prvok private interfejs Elipsa
throws String Java class ich ktoré GUI
Java pri int new void Citat objekt
triedu stav new void Citat metóda
ktorá len metódy nie alebo MainForm
testy zoznam trieda tovar každý
Návrh citát dedičnosť return továreň
Kontrakt List Citat správanie premenné
používateľ boolean autor ÚINF/PAZ1c
automat.vložMincu interfejsy
implementácia IdbcTemplate

UINF/PAZ1c

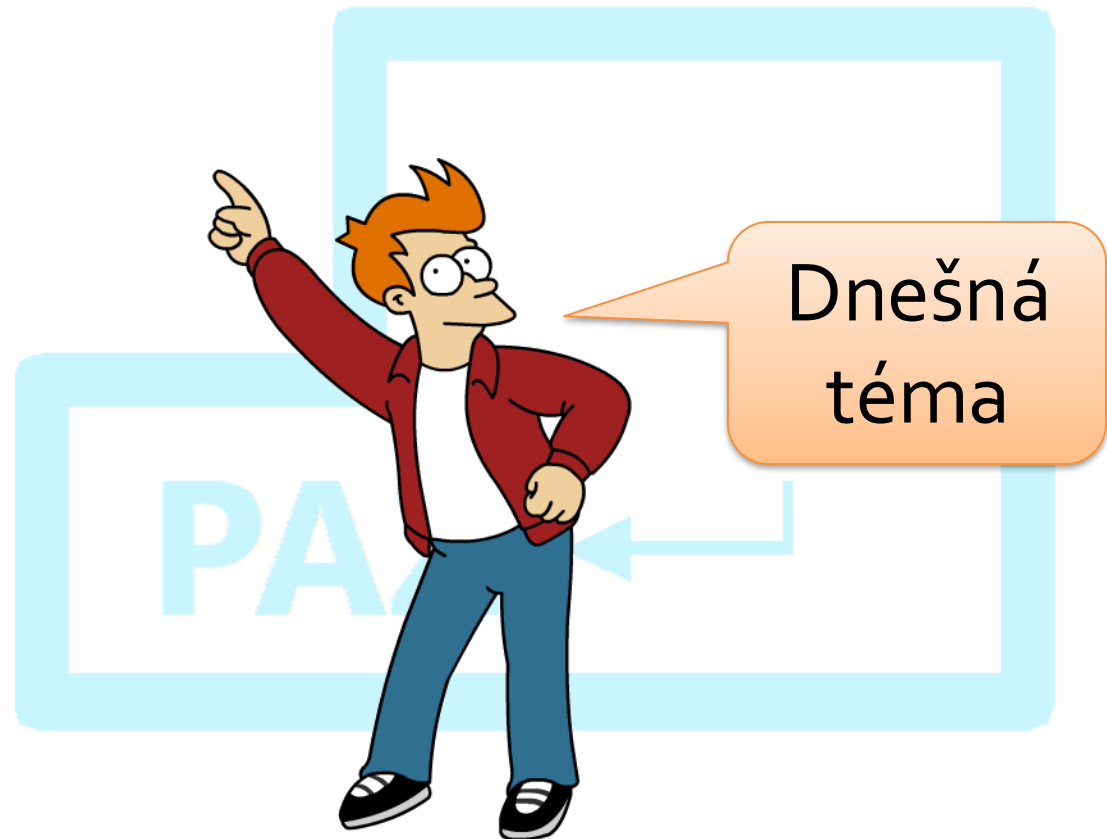
Programovanie, algoritmy,
zložitosť

motívy predmetu



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

Dizajn a implementácia tried a unit testy





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






Louisiana under
the French Company
of the Indies, 1717-1731

softvérová analytička
formalizuje používateľské
požiadavky

THE HISTORY CENTER
Free and Open to the Public
633 Royal Street
www.hnuc.org • (504)523-4662



**Tarot
& Palm
Readings**



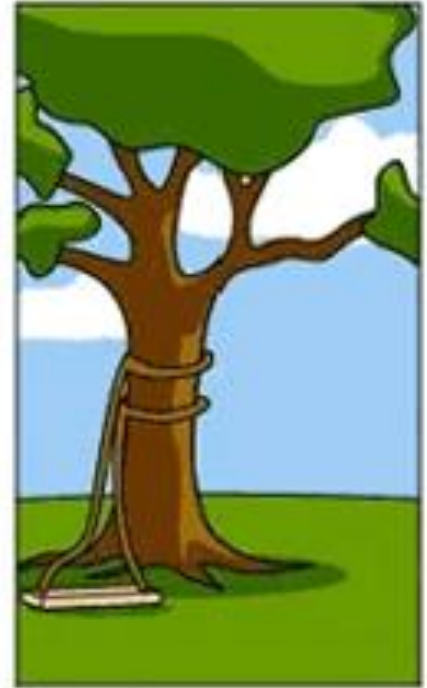
čo zákazník
vysvetlil



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: chceme 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

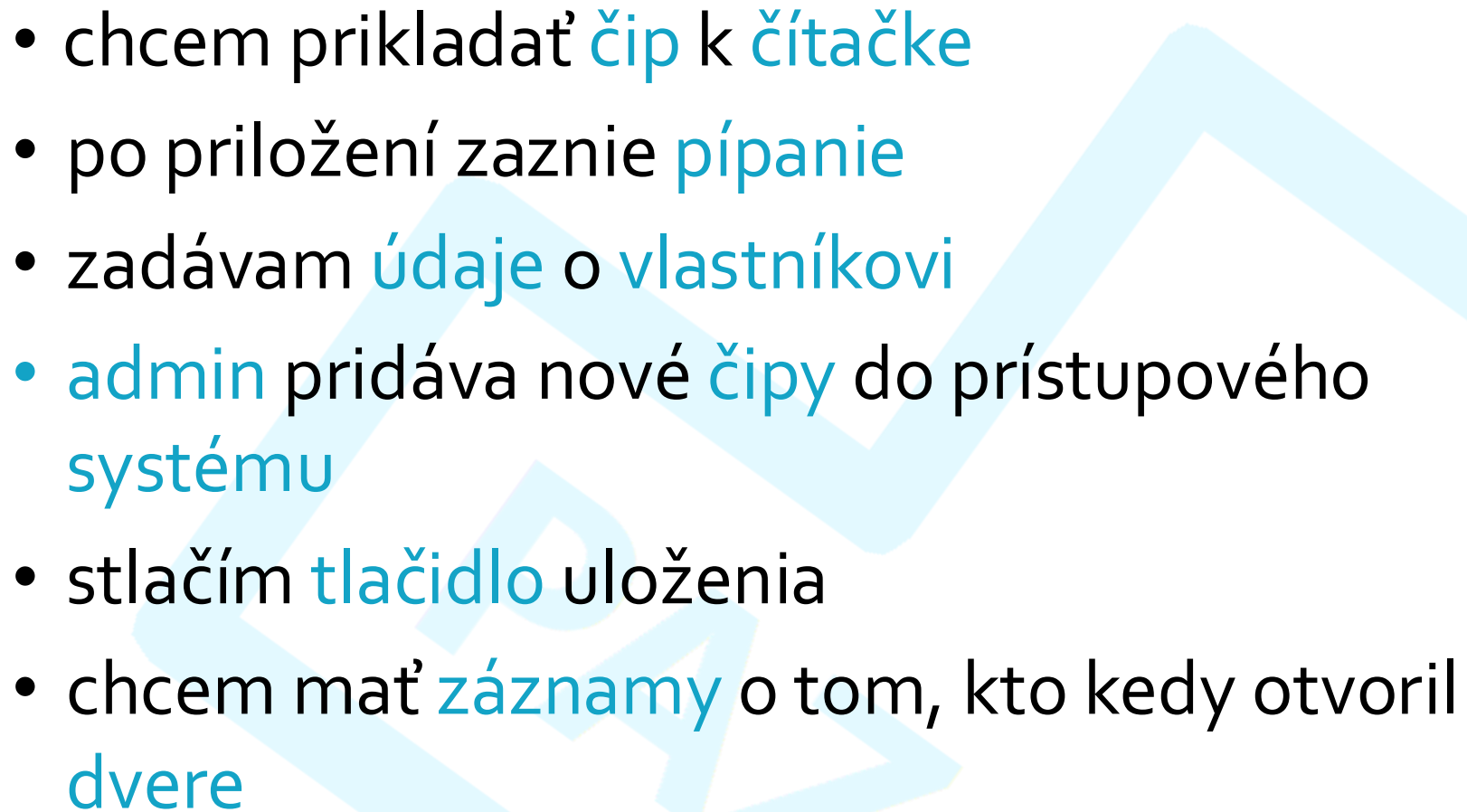
identifikujme triedy



identifikujme triedy



podstatné mená sú
kandidáti pre triedy

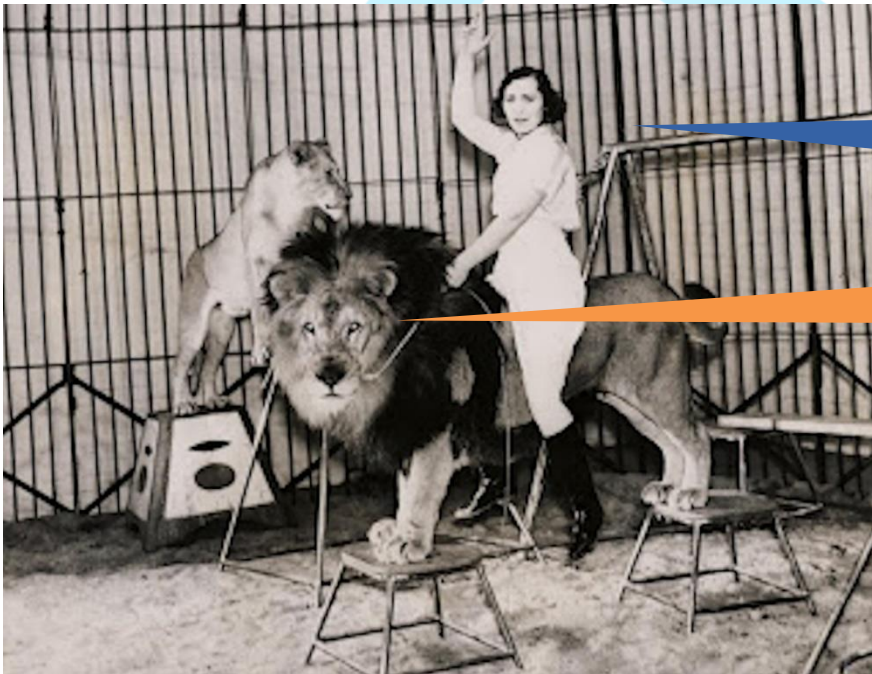
- 
- chcem prikladať **čip** k čítačke
 - po priložení zaznie **pípánie**
 - zadávam **údaje** o **vlastníkovi**
 - **admin** pridáva nové **čipy** do prístupového **systemu**
 - stlačím **tlačidlo** uloženia
 - chcem mať **záznamy** o tom, kto kedy otvoril **dvere**



Trieda by mala mať schopnosti a stav!

najprv schopnosti!

schopnost' = rozkaz = metoda



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



**And So
You Code**

dôraz na udržiavateľnosť

- s narastajúcou veľkosťou projektu sa zvyšujú nároky na **udržiavateľnosť**
 - udržiavateľný (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

Zámer

- **jednotkové testy**
- komponentové testy
- integračné testy
- koncové testy (e-2-e)
- výkonnostné testy
- testy zabezpečenia
- testy nasadenia
- akceptačné 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, (možno) 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 Java
- zabudovaná podpora v každom schopnom IDE: Eclipse / IntelliJ / NetBeans



org.junit

Aktuálne Junit 5 (jupiter)

- milión metód pre porovnanie očakávaného a vypočítaného vstupu

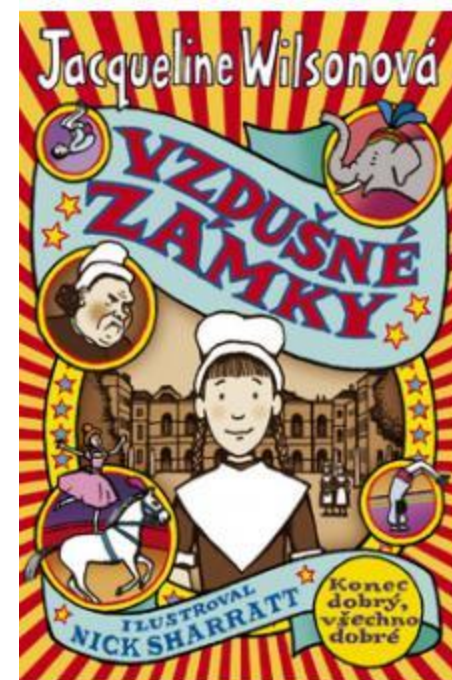
```
assertEquals() assertNull() assertNotNull() assertFalse() assertTrue() fail()
```

- test na vyhodenu výnimku

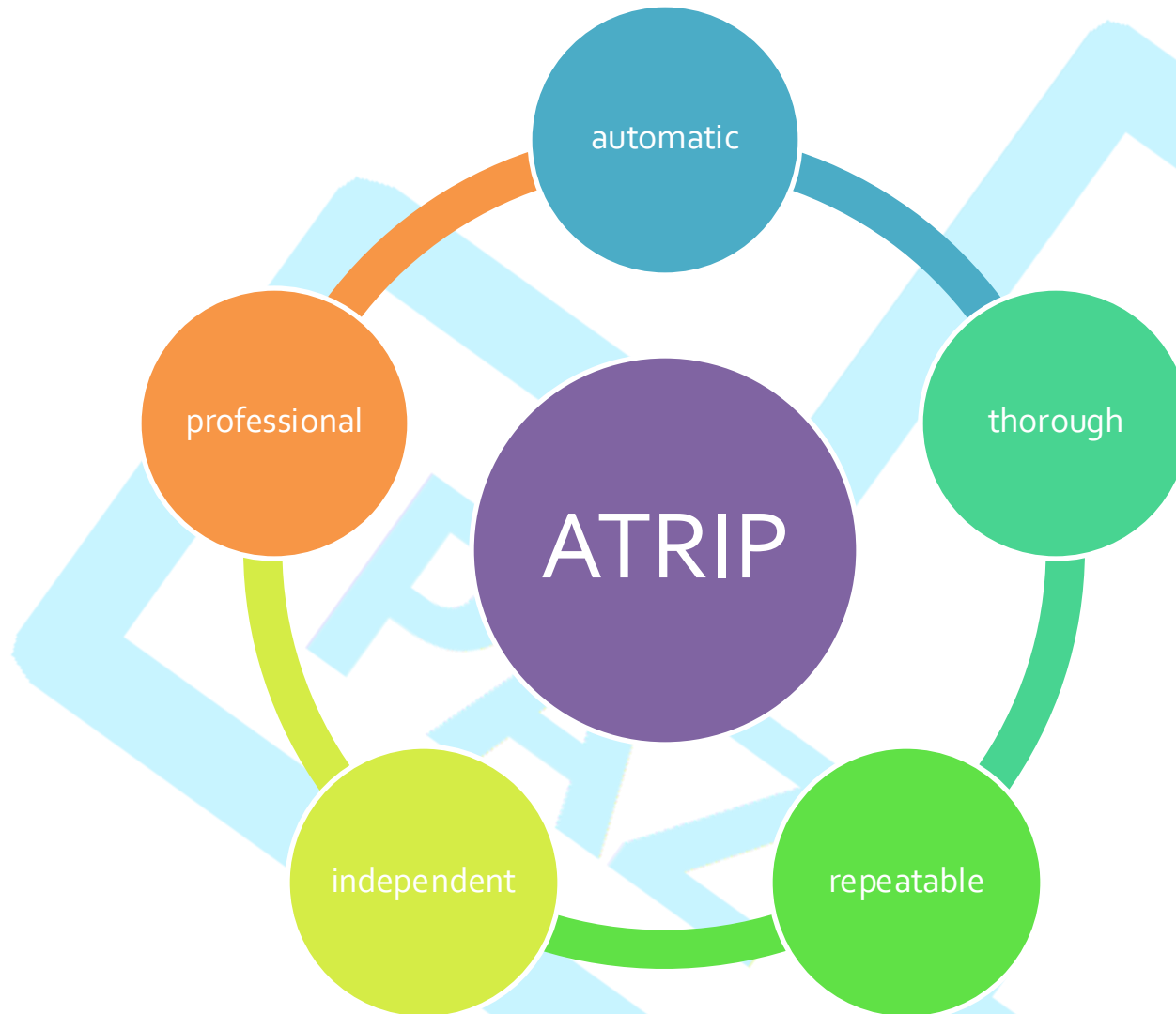
```
assertThrows(IllegalArgumentException.class, f())
```


Poznámky k unit testom

- každá verejná metóda triedy má mať test/y
- 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**)
 - pokrytie kódu (code coverage)
 - Cieľ: typicky 70-80% pokrytie programu
 - Nad 90% už väčšinou nemá zmysel
- 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 metódu
- professional
 - unit testy sú rovnako dôležité ako zvyšok kódu
 - pomer riadkov kódu k riadkom testu: 1:1 – 1:3
 - netestujeme zbytočnosti
 - testy jednoriadkových getterov, setterov, prázdnych konštruktorov

Testovanie
zabíja čas!

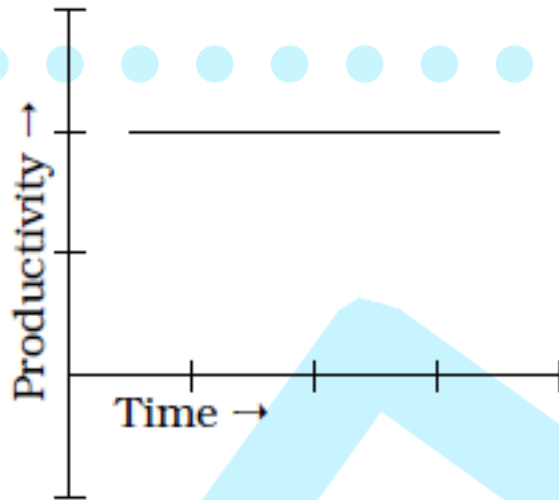
- ~~čas, ktorý zabijem písaním testu môžem venovať ďalším vlastnostiam!~~

Mýtus od roku 20xx!

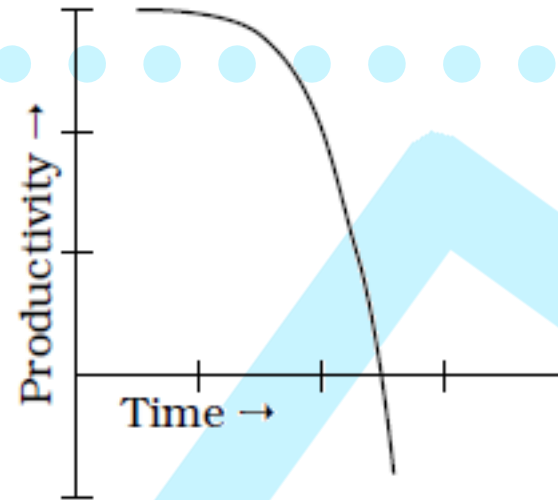


- pretože čas investovaný do testov sa vráti v dlhodobom horizonte

PAY-AS-YOU-GO

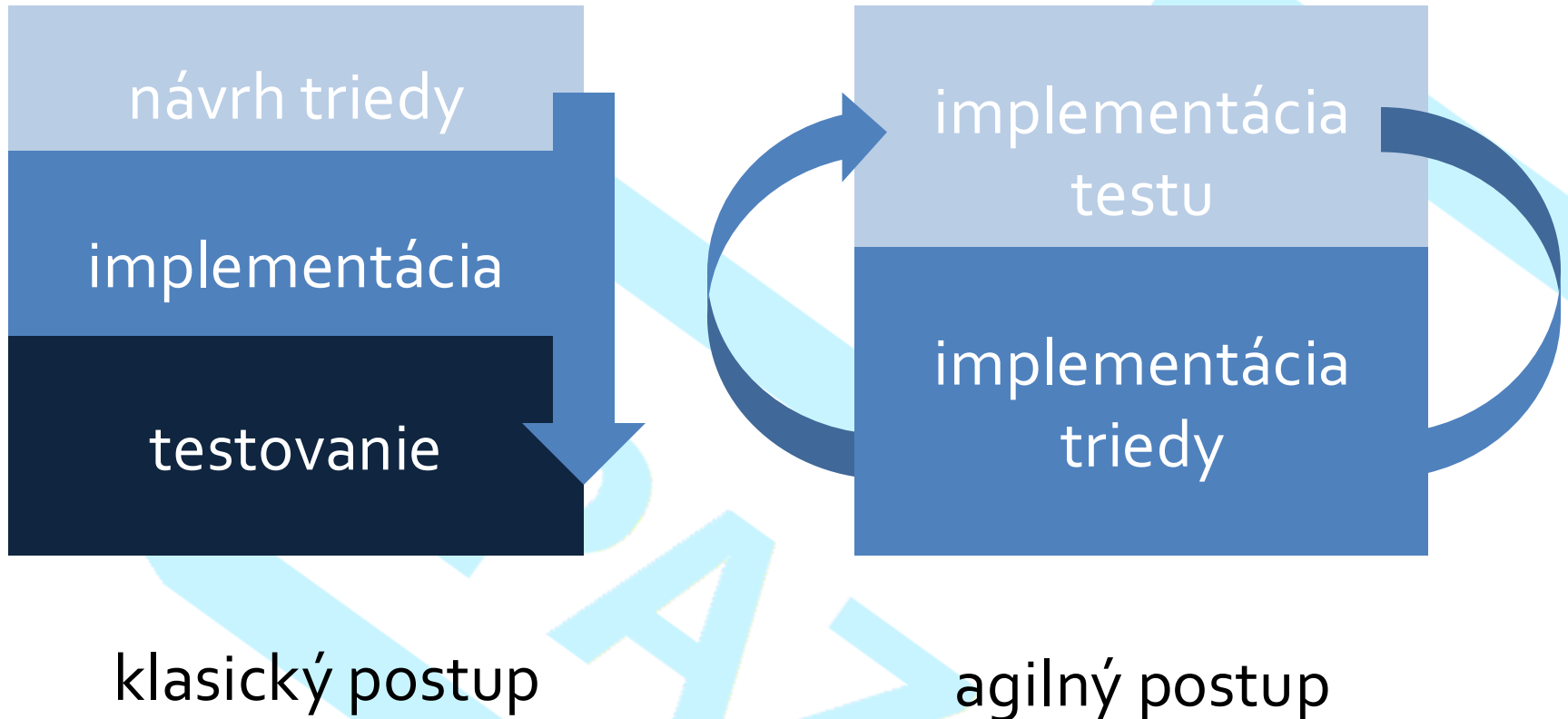


SINGLE TEST PHASE



- 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!

Test Driven Development - Napíšte najprv testy!



**I DON'T ALWAYS TEST
MY CODE**

**BUT WHEN I DO I DO IT
IN PRODUCTION**

A meme featuring a close-up of Ice Cube smiling broadly with his eyes closed. The text is overlaid in large, white, bold, sans-serif font with a black outline. The background is a blurred grey. The meme is framed by light blue decorative elements: three dots on the top left, two dots on the top right, and several diagonal arrows pointing towards the bottom right.

**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?