TVORBA MOBILNÍCH APLIKACÍ

IOS TESTOVÁNÍ

KDO JSEM

- Ing. Ondřej Hanák
- FIT VUT
- 2010 diplomka v C# pro WM
- iOS aplikace od 2012, předtím weby
- freelancer, vlastní aplikace, UK studio, US startup
- oh@ondrejhanak.cz

OBSAH

- teorie
  - proč, co, jak
  - budu se opakovat
- ukázky
  - Swift
  - live coding

PROČ TESTOVAT

- proč mají auta brzdy?

PROČ TESTOVAT

- testy máme, abychom mohli vyvíjet rychleji
- dovolují nám dělat sebevědomé změny
- ukazují, že kód dělá, co má
- počáteční investice se vrátí časem i s úroky

PROČ TESTOVAT

- hlídání regresí
- kontrola funkce dle očekávání, nikoliv nepřítomnost bugů
- “Quality Assurance, not Quality Insertion.”
- netestujeme privátní kód
- code coverage

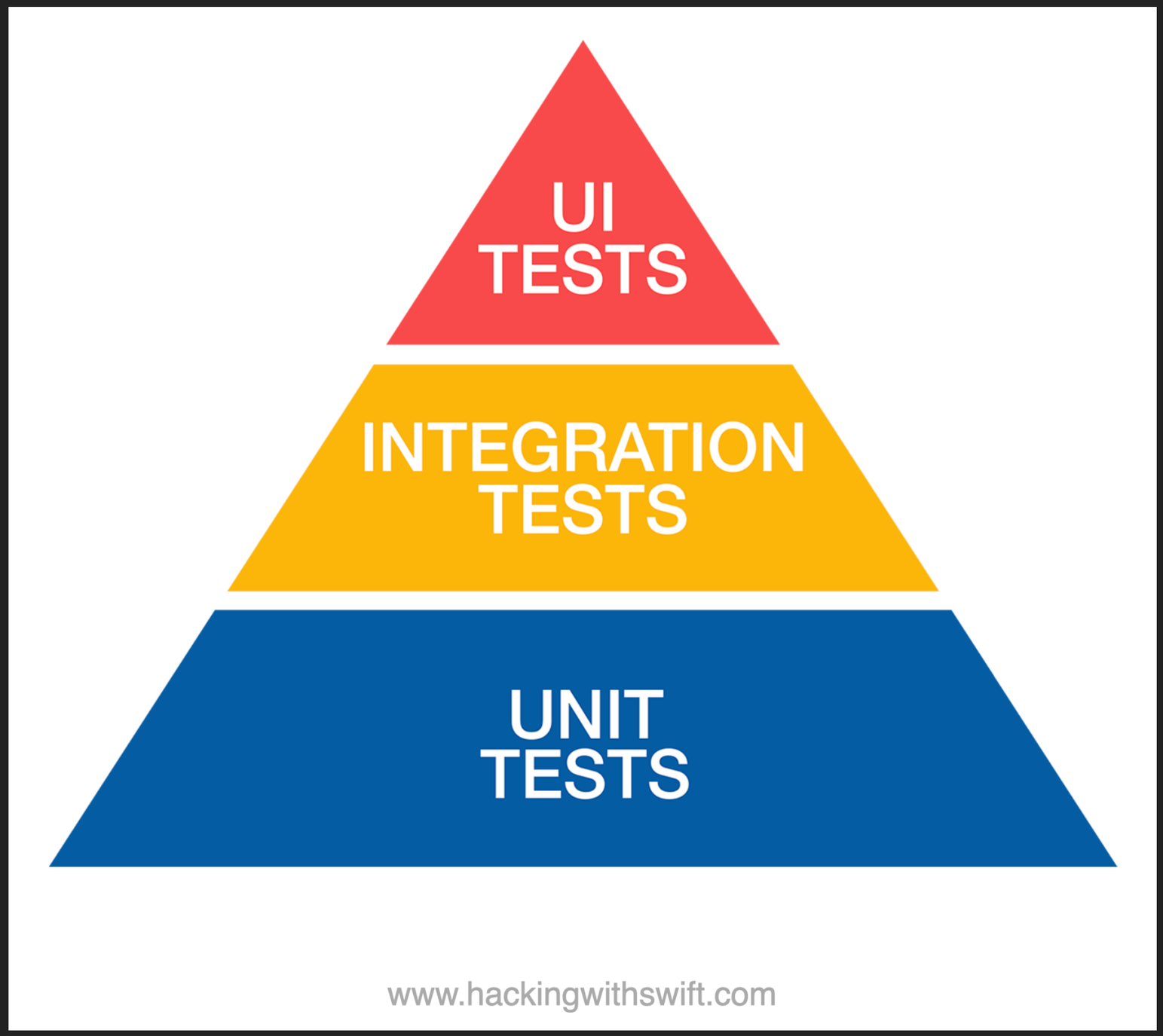
DRUHY TESTOVÁNÍ

- unit testy
- integrační testy
- UI testy
- zátěžové / výkonnostní testy
- smoke testy

ZPŮSOB PROVÁDĚNÍ

- automaticky (CI)
- manuálně
- hallway testing
- dogfooding
- beta testeři

TESTOVACÍ PYRAMIDA



FIRST

- Fast
- Isolated
- Repeatable
- Self-verifying
- Timely

FIRST

- **Fast**
  - musí jich proběhnout stovky nebo i tisíce za sekundu
  - nesmí zdržovat a tím snižovat ochotu je spouštět

FIRST

- **Repeatable**
  - pro stejný vstup musí dát vždy stejný výstup
  - nezávisle na počtu opakování nebo fázi měsíce
  - flaky testy

FIRST

- **Self-verifying**
  - musí být jasné, jestli prochází nebo ne
  - žádný prostor pro interpretaci

FIRST

- **Timely**
  - měly by být psány současně s kódem, který testují
  - dokud to má vývojář v hlavě
- TDD

UNIT TESTY

- samostatné jednotky izolovaně
  - riziko vzájemného ovlivňování jednotek
  - riziko nejasnosti důvodu selhání
- jakmile se testuje více jednotek současně, nejde o unit test, ale integrační test
- zdola nahoru

UNIT TESTY

- fast - ano
- isolated - ano
- repeatable - ano
- self-verifying - ano
- timely - ideálně ano

INTEGRAČNÍ TESTY

- testují vzájemnou součinnost více jednotek naráz
- fast: ani ne, protože dělají reálnou práci (např. přístup na síť)
- isolated: ano, ve smyslu ostatních jednotek než zahrnutých v testu

INTEGRAČNÍ TESTY

- repeatable: ano
- self-veryfing: nemusí , někdy může být potřeba ověřit výsledek manuálně
- timely: ne, není obvyklé, že by se psaly v době tvorby jednotlivých jednotek

2 UNIT TESTY, 0 INTEGRAČNÍCH

2 UNIT TESTY, 0 INTEGRAČNÍCH

UI TESTING

- interakce s reálnou aplikací a kontrola UI
- manuálně i automaticky
- testovací prostředí ovládá app v simulátoru a vyhodnocuje dotazy na stav UI
- nemá ponětí o kódu a vnitřní struktuře - blackbox

UI TESTING

- nejlépe odpovídá tomu, co dělá koncový uživatel
- technologie není tak vyzrálá jako pro unit testy
- složité zjišťování proč přesně test selhal
- změny v UI typicky rozbijí testy
- jsou tedy pomalé jak na tvorbu, tak na běh

UI TESTING

- fast - ne
- isolated - ne
- repeatable - záleží
- self-verifying - záleží
- timely - ne

TO SI NEMŮŽEME DOVOLIT

- klient za testy nezaplatí
- jsme startup, nemáme peníze na testy
- musíme produkt rychle vydat, nemáme čas na testy

TO SI NEMŮŽEME DOVOLIT

- Když klient nechce platit za testy, bude chtít za bugfixy?
- Klienta nezajímá, jestli používáme MVC nebo MVVM
- Stejně by ho nemělo zajímat zda testujeme, to je naše zodpovědnost a reputace
- "Testy nejsou práce navíc. Je to práce namíň."

QA

- vývojář zde není tester
- jsou ale na stejné lodi
- “We don’t break the software, we break illusions about the software.”

NÁZVOSLOVÍ

- test case
- test suite
- SUT
- assertions

ŽIVOTNÍ CYKLUS TESTU

- třída XCTestCase
- testSomethingSomehow()
- test_Something_Somehow()
- DISABLED_test_Something_somehow()
- setUp() a tearDown()
- continueAfterFailure

STRUKTURA TESTU

- arrange - act - assert paradigma
- given - when - then
- příprava vstupu
- provedení akce
- ověření výstupu

ASSERTIONS

- XCTAssertEqual(output, expectation)
- XCTAssertTrue(output == expectation)
- XCTAssertTrue(result)
- XCTAssertFailure()
- není XCTAssertSuccess()
- vlastní zpráva k assertu

REFACTORING

- změna vnitřní struktury bez změny vnějšího chování
- vnější chování kontrolují testy
- bez nich jde o přepisování

TESTOVATELNÝ KÓD

- snaha o omezení stavových informací
- jistota, že výsledek testu závisí jen na vstupech
- problém skrytých závislostí
- design rozhraní pro interfejsy, ne implementace
- využití dále při mockingu

DEPENDENCY INJECTION

- objekty dostávají data a závislosti, se kterými pracují
- nezískávají je sami z prostředí
- “Tell, don’t ask.”
- Constructor / property / closure injection

MOCKING

- izolace
- nahrazení závislostí dvojníky (test doubles)
- jako kaskadéři u filmu - jiné dovednosti,
  ale vizáž původního herce
- různé varianty dle míry napodobení

MOCKING

- **dummy**: objekt nic nedělá, ale je potřeba aby se kód vůbec zkompiloval, metody jsou prázdné nebo co možná nejprázdnější
- **stub**: objekt vracející fixní data
- **mock**: navíc sleduje, zda a jak byly části mocku volány pro pozdější ověření v testu

MOCKING

- souhrnně se používá pojem mocking
- **full mock**: nový objekt pro potřebu testu
- **partial mock**: subclass existujícího objektu a překrytí vybraných metod pro potřeby testu
- zřejmě nejnáročnější část psaní testů
- snaha mockovat co nejmenší podmnožinu funkcí, abychom dosáhli požadovaného chování

MOCKING SÍŤOVÁNÍ

- typický příklad pomalé a nespolehlivé závislosti
- problém s obsahem vzdálené databáze při opakování
- z principu porušuje FIRST
- řešení skrz napojení na subsystém zpracování HTTP požadavků, odchycení správného požadavku a vrácení vhodných dat
- funguje následně i offline

ZDROJE INFORMACÍ

- Testing Swift (Paul Hudson)
- Google
- libovolná kniha o xUnit frameworku / platformě

KONEC