



Mostoha Roland

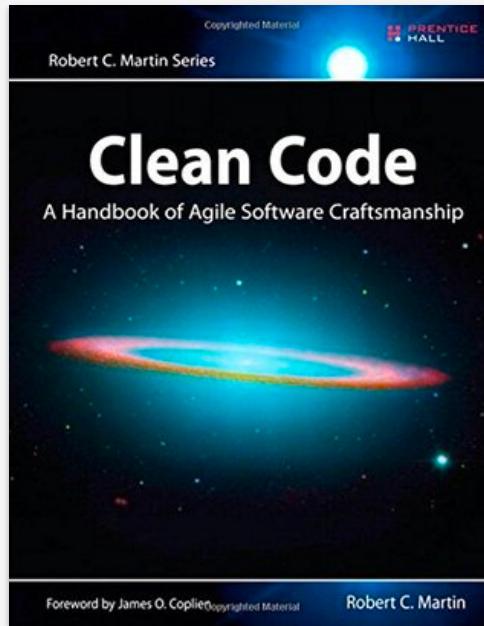
SquadLead @ AutSoft

Tesztelési lehetőségek, Unit és UI tesztek



Tesztelési lehetőségek, Unit és UI tesztek

Automatizált tesztelés



- Története a 60-as évekre nyúlik vissza
- 2000-es években lett igazán népszerű
 - Agilis szoftverfejlesztés
 - Clean Code elvek (Robert c. Martin)
 - Test Driven Development (Kent Beck)
- Sok cégnél ez a de-facto standard megközelítés



Tesztelési lehetőségek, Unit és UI tesztek

Automatizált tesztelés



- Pár éve került előtérbe
- Szükség van egyedi megoldásokra, speciális eszközökre
- Megjelentek hivatalos architekturális minták
- Tesztek nyílt forráskódú Google alkalmazásokban



Mit vihetünk hazá?

- Egyedülálló, értékes tudás a piacon
 - Mobilos körökben még nem terjedt el annyira
 - Sok cégnél még nincs kialakulva az automatizált tesztelési kultúra
- Saját cégünkönél úttörők lehetünk ☺
- Összefüggések megértése az architektúrák, Clean Code elvek kapcsán
- Stabil tudás, hogy magabiztosan el tudunk kezdeni tesztelni
 - Az automatizált tesztelés során leggyakrabban előforduló problémákat vesézzük

Agenda



1

2

The PIN Project

Tesztelési alapelvek

- Miért teszteljünk?
- Teszt típusok
- Tesztelési szintek

Unit tesztelés

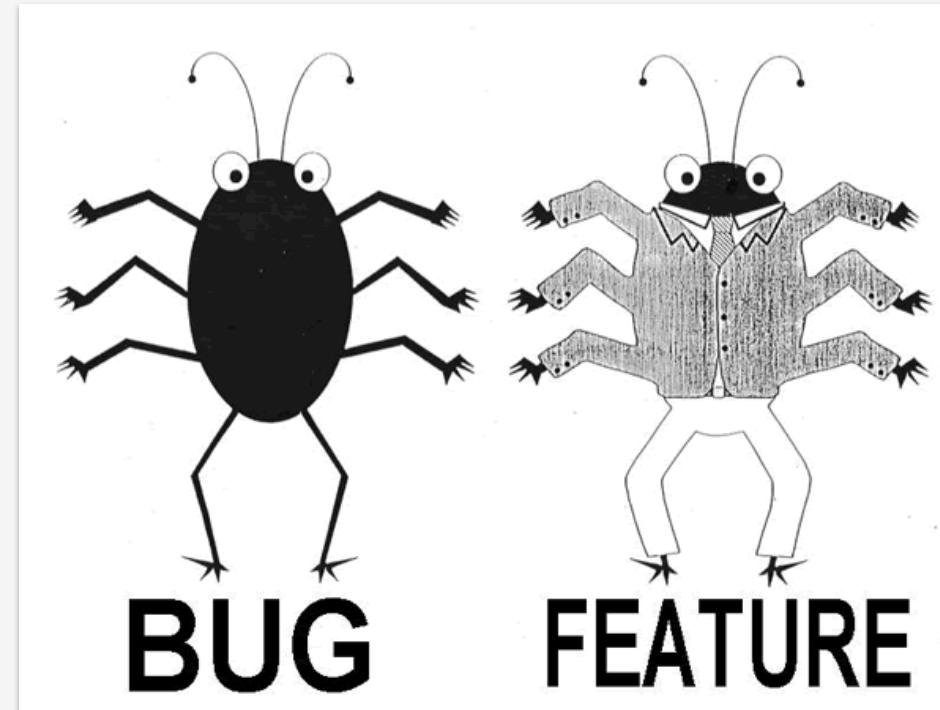
- Kihívások Androidon
- Módszerek, eszközök
- Architektúra

Instrumentation tesztelés

- Módszerek, eszközök
- Kihívások, problémák

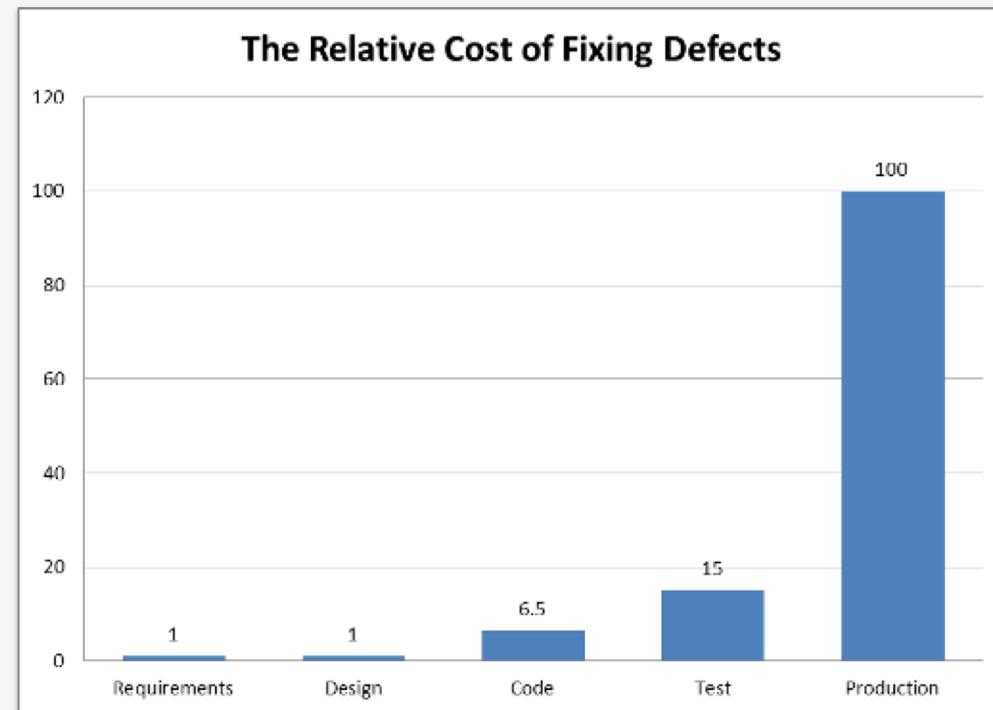
Miért teszteljünk?

- A fejlesztői hibák elkerülése



Miért teszteljünk?

- A költség csökkentése



Forrás: IBM Systems Sciences Institute

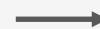


Miért teszteljünk?

- A tesztek dokumentálják az elvárt funkcionálitást
- Specifikációként tekinthetünk rájuk

User Story

"Miután a felhasználó helyes PIN kódot írt be, átnavigáljuk a Főoldalra."



```
@Test
fun givenValidPin_whenWeSubmit_thenMainScreenShouldShow ... {
    // Given
    val pin = "12345"
    // When
    onView(withId(R.id.input_pin))
        .perform(typeText(pin))
        .perform(closeSoftKeyboard())
    onView(withId(R.id.btn_submit))
        .perform(click());
    // Then
    onView(withText(R.string.home_title))
        .check(matches(isDisplayed())))
}
```

Miért teszteljünk?

- A tesztelés megkövetel bizonyos szintű kód-minőséget
- Elősegíti a karbantartható kódot





Tesztelési alapelvek

Miért teszteljünk?



- minden egyes megírt teszt pozitív visszajelzés arról, hogy helyes, amit csinálunk
- Megbízunk a saját kódunkban
 - Szívesebben fejlesztünk bele legközelebb
 - Kockázat-mentes refactor
 - Kockázat-mentes funkcióbővítés



Miért ne teszteljünk?

- Túl sok időt emészt fel
- A megrendelő nem fizeti ki
- Nem éri meg
- Nem tudom, mit teszteljek
- Tökéletes és hibátlan kódot írok
- Nehéz



Miért ne teszteljünk?

- ~~Túl sok időt emészt fel – A ráfordítást visszakapjuk a hibajavítás során~~
- A megrendelő nem fizeti ki
- Nem éri meg
- Nem tudom, mit teszteljek
- Tökéletes és hibátlan kódot írok
- Nehéz



Miért ne teszteljünk?

- Túl sok időt emészt fel – A ráfordítást visszakapjuk a hibajavítás során
- A megrendelő nem fizeti ki – A hibajavítást kifizeti
- Nem éri meg
- Nem tudom, mit teszteljek
- Tökéletes és hibátlan kódot írok
- Nehéz



Miért ne teszteljünk?

- ~~Túl sok időt emészt fel~~ – A ráfordítást visszakapjuk a hibajavítás során
- ~~A megrendelő nem fizeti ki~~ – A hibajavítást kifizeti
- ~~Nem éri meg~~
 - Nem tudom, mit teszteljek
 - Tökéletes és hibátlan kódot írok
 - Nehéz



Miért ne teszteljünk?

- Túl sok időt emészt fel – A ráfordítást visszakapjuk a hibajavítás során
- A megrendelő nem fizeti ki – A hibajavítást kifizeti
- Nem éri meg
- Nem tudom, mit teszteljek – Mindent, ami a specifikációban van
- Tökéletes és hibátlan kódot írok
- Nehéz



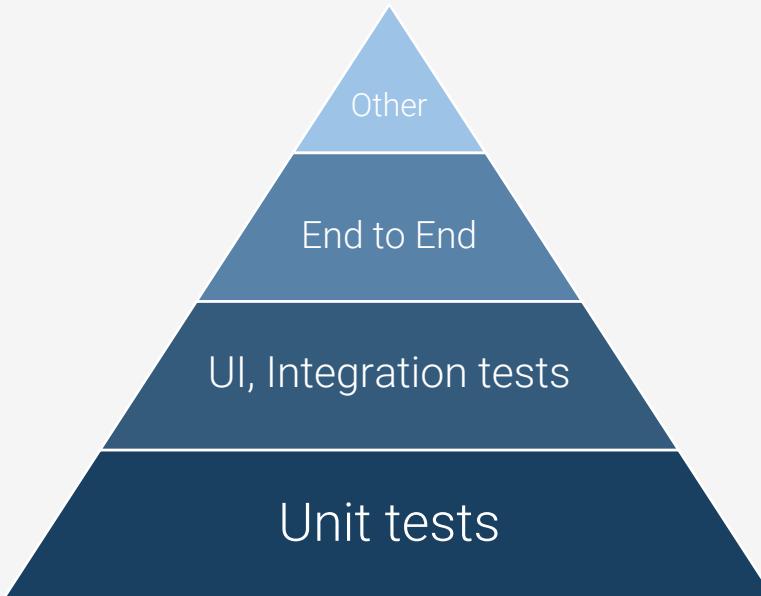
Miért ne teszteljünk?

- ~~Túl sok időt emészt fel~~ – A ráfordítást visszakapjuk a hibajavítás során
- ~~A megrendelő nem fizeti ki~~ – A hibajavítást kifizeti
- ~~Nem éri meg~~
- ~~Nem tudom, mit teszteljek~~ – Mindent, ami a specifikációban van
- ~~Tökéletes és hibátlan kódot írok~~
- Nehéz



Miért ne teszteljünk?

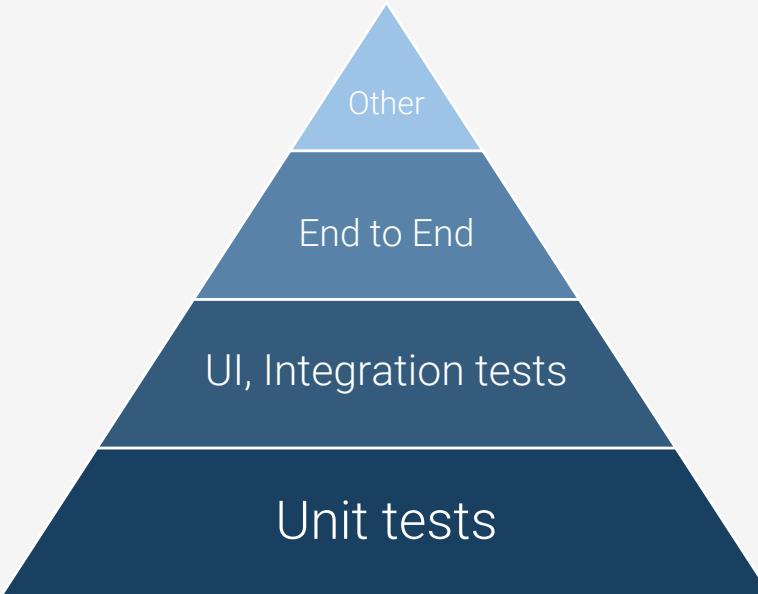
- ~~Túl sok időt emészt fel~~ – A ráfordítást visszakapjuk a hibajavítás során
- ~~A megrendelő nem fizeti ki~~ – A hibajavítást kifizeti
- ~~Nem éri meg~~
- ~~Nem tudom, mit teszteljek~~ – Mindent, ami a specifikációban van
- ~~Tökéletes és hibátlan kódot írok~~
- ~~Nehéz~~ – De megéri



Tesztelési alapelvek

Általános tesztelési szintek

- **Unit:** Osztály, metódus szintű
- **UI, Integration:** Több modulon átívelő tesztek
- **End to End:** Aktív hálózat, adatbázis, szenzorok stb.
- **Other:** Pl. UIAutomator, Monkey-runner tesztek



Tesztelési alapelvek

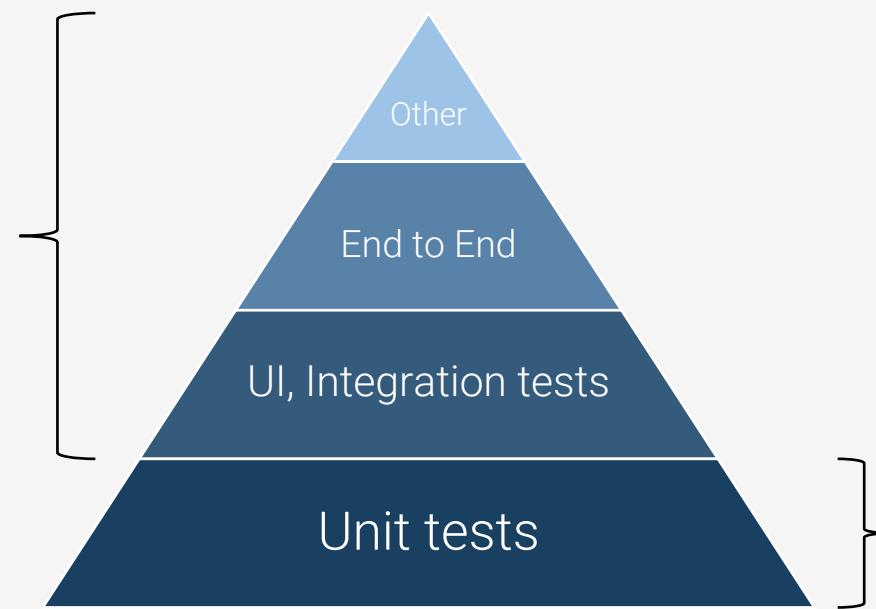
A piramis aspektusai

- Inkább fontossági mint mennyiségi aspektus
 - Sokan rosszul értelmezik és csak Unit tesztet írnak
 - [I'm so sick of the testing pyramid](#)
- Hierarchikus aspektus
 - minden szint az alatta lévőre épít

Tesz típusok Androidon

Instrumentation tesztek

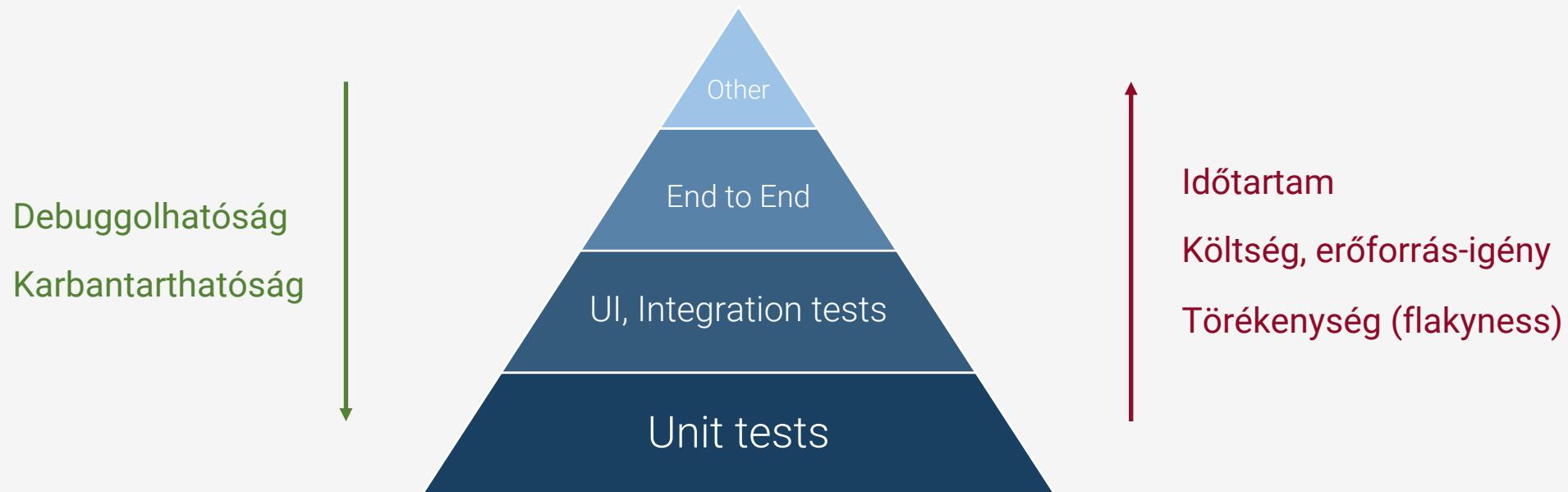
- Valós eszközön futnak
- Valós Android kontextus



JUnit tesztek

- JVM-ben futnak
- Android kontextus mockolva

A tesztelési piramis



Agenda



1

2

The PIN Project

Tesztelési alapelvek

- Miért teszteljünk?
- Teszt típusok
- Tesztelési szintek

Unit tesztelés

- Kihívások Androidon
- Módszerek, eszközök
- Architektúra

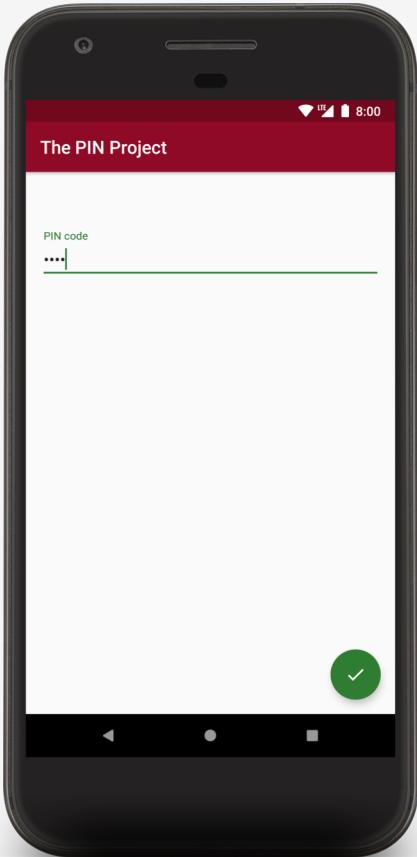
Instrumentation tesztelés

- Módszerek, eszközök
- Kihívások, problémák



Effective Automated Testing

The PIN Project



- Master: Alkalmazás naív implementációja
- Step 1: UI tesztek Espresso segítségével
- Step 2: Unit tesztek a Validator funkcióira
- Step 3: Szerver oldali validáció implementációja
- Step 4: Dependency Inversion Principle + Dependency
- Step 5: Model-View-Presenter megvalósítása

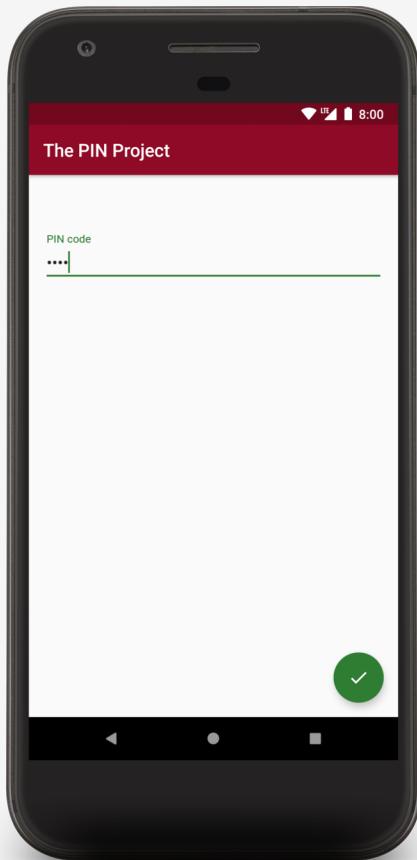


<https://github.com/RolandMostoha/the-pin-project>



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Készítsünk egy PIN kód bekérő alkalmazást!

- A beviteli mező csak számokat tartalmazhat
- Helyes PIN megadása után átnavigáljuk a főoldalra
- Helytelen PIN megadását üzenetben jelezzük
- A mesterkulcs 7878



<https://github.com/RolandMostoha/the-pin-project/tree/master>



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Készítsünk automatizált teszteket az alkalmazáshoz!

- Milyen teszttel kezdjük?
- Milyen támogatás van rá Android platformon?
- Hogyan kezdjük, milyen módon változtathatunk a kódon?



JUnit

Unit tesztelés Androidon

JUnit

- Tisztán a JVM-ben fut
 - Nincs Android kontextus
 - Nincs aktív hálózat, adatbázis, kamera, szenzorok
 - Gyors lefutás (pár ms)
 - Könnyen debuggolható
 - Olcsó
- Rákényszerít, hogy a külső függőségeket leválasszuk
 - Karbantartható
 - Clean Code elveket segít betartani, javítja a kód minőségét



Instrumentation tesztelés Androidon

Instrumentation tesztek



- Valós eszközön/emulátoron fut
 - Nem feltétlenül kell bekapcsolt képernyővel futnia
- Android kontextussal rendelkezik
- Lehet aktív hálózat, adatbázis, kamera, szenzorok
- Lassú lefutás (másodpercek)
- Könnyen debuggolható
- Nem követeli meg a karbantarthatóságot
- Viszonylag drága – valós eszközpark

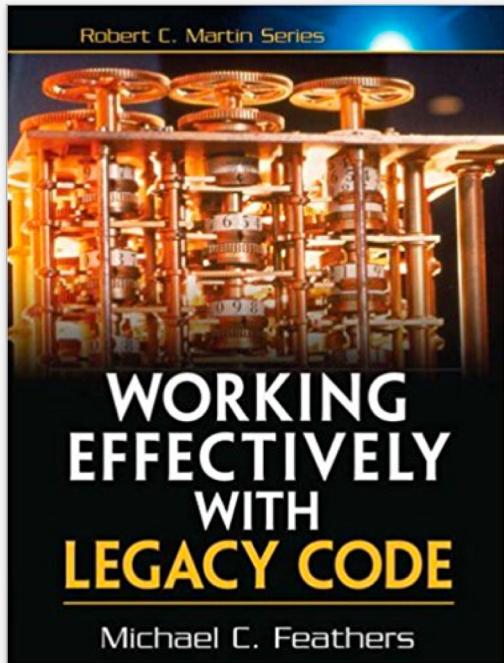
Felületi vagy UI tesztek



- Valós eszközön/emulátoron fut
 - Bekapcsolt képernyővel kell futnia
- Android kontextussal rendelkezik
- Lehet aktív hálózat, adatbázis, kamera, szenzorok
- Lassú lefutás (másodpercek)
- Nehezen debuggolható
- Nem követeli meg a karbantarthatóságot
- Drága – valós eszközpark, bekapcsolt képernyővel



The PIN Project



Milyen teszttel kezdjük?

- minden rendszer "Legacy", amihez nem tartozik teszt
- A kód változtatása nélkül szeretnénk megbizonyosodni a viselkedésről
- A legacy rendszerek feltérképezésére jó módszer magas szintű tesztek írása
 - Ezekkel képet kapunk a rendszer működéséről
 - Bebiztosítjuk magunkat, hogy nem törjük el a működést
 - Lehetséges kockázat-mentes refactor, funkcióbővítés



Instrumentation tesztelés Androidon

Espresso tesztek



- UI-teszt keretrendszer
- Egyszerű, könnyen kezelhető
- Adott View elemen végezhetünk
 - Ellenőrzéseket
 - Akciókat
 - Click
 - Swipe
 - Input mező kitöltés

<https://developer.android.com/training/testing/espresso/cheat-sheet>



Instrumentation tesztelés Androidon

Espresso tesztek



- Támogat WebView-t
- AdapterView, RecyclerView, DatePicker stb.
- Szinkronizációt támogat
 - AsyncTask, MessageQueue alapértelmezetten
 - Egyedi esetben IdlingResource
- Egyedi Matcher-ek, Action-ök készíthetőek vele



Instrumentation tesztelés Androidon

Espresso tesztek



```
@Test
fun givenValidPin_whenWeSubmit_thenMainScreenShouldShown ... {
    // Given
    val pin = "7878"
    // When
    onView(withId(R.id.input_pin))
        .perform(typeText(pin))
        .perform(closeSoftKeyboard())
    onView(withId(R.id.btn_submit))
        .perform(click());
    // Then
    onView(withText(R.string.home_title))
        .check(matches(isDisplayed()))
}
```



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



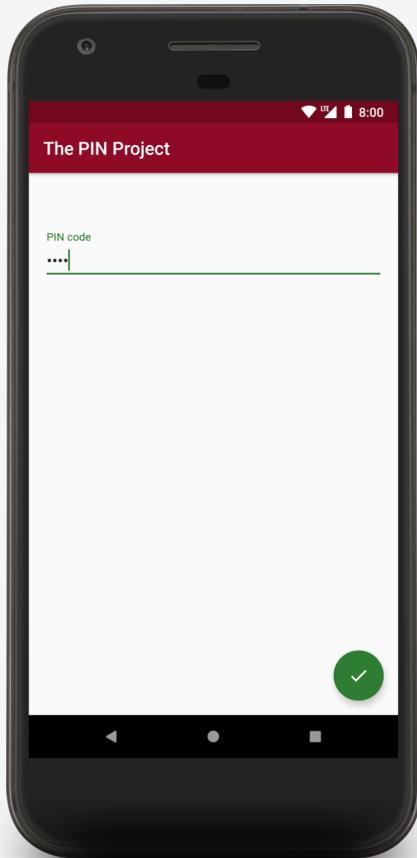
Készítsünk UI teszteket az alkalmazáshoz!

- A beviteli mező csak számokat tartalmazhat
- Helyes PIN megadása után átnavigáljuk a főoldalra
- Helytelen PIN megadását üzenetben jelezzük
- A mesterkulcs 7878



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



- ✓ Megbizonyosadtunk az alkalmazás működéséről
- ✓ A tesztek garantálják, hogy a működést nem rontjuk el
- ✓ Lehetőség van kockázat mentes refaktorra
- ✓ Lehetőség van kockázat mentes funkció bővítésre



<https://github.com/RolandMostoha/the-pin-project/tree/step-1-UI-tests>



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project

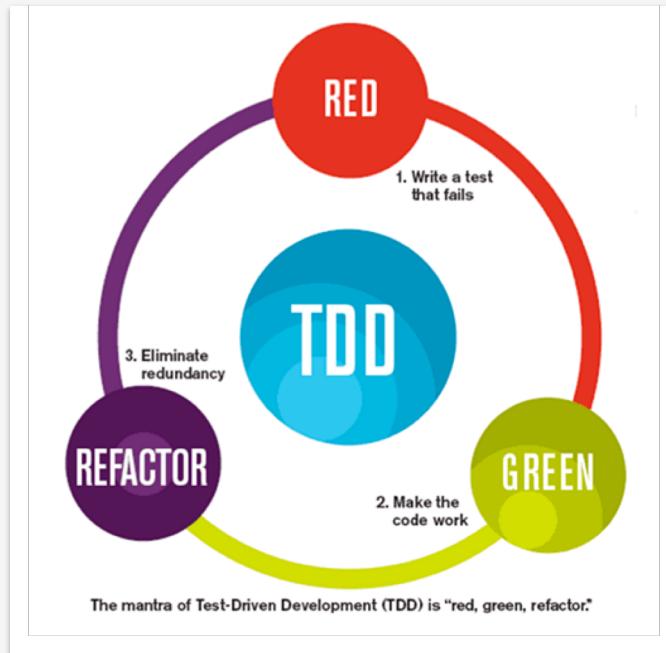


Javítsunk az alkalmazás kódján, készítsünk Unit teszteket!



Tesztelési lehetőségek, Unit és UI tesztek

Test Driven Development



1. Írunk egy bukó tesztet az adott funkcióra
2. Írunk pont annyi kódot, amivel a tesztet megjavítjuk
3. Refaktorálunk, javítsunk a kód minőségén
4. Ismételjük a folyamatot

Tesztelési lehetőségek, Unit és UI tesztek

Tesztvezérelt fejlesztés előnyei

- Biztosan írunk tesztet 😊
- Biztosan arra írunk tesztet, amire szerethnénk
- Biztosan refaktorálunk
- Segítjük a kódminőség javítását

Hagyományos megközelítés



Test Driven Development

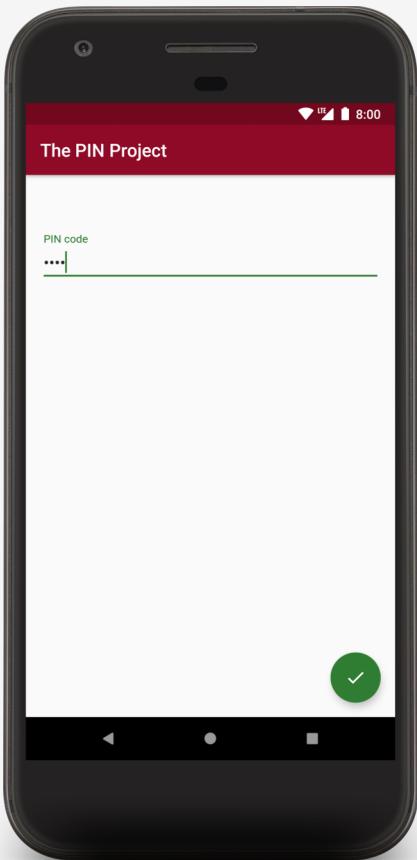


Alkalmazzuk a TDD-t a Unit tesztek készítésekor!



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



- ✓ Lefedtük Unit tesztekkel is a funkciókat
- ✓ Átláthatóbb, tisztább kódot kaptunk a refaktor miatt
- ✓ Az osztályok felelősségei jobban elkülönülnek
- ✓ A tesztek futásának idejét levittük 25ms-re



<https://github.com/RolandMostoha/the-pin-project/tree/step-2-Unit-tests-TDD>



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Validáljuk a beütött PIN kódot szerver oldalon is!

- Szimuláljunk egy szerver oldali kérést amely
 - Visszatér egy random TRUE vagy FALSE értékkel
 - 500ms-1000ms random válaszidővel rendelkezik
- Ha szerver oldali validáció is helyes, akkor navigálunk
- Ha helytelen, akkor üzenetben jelezzük

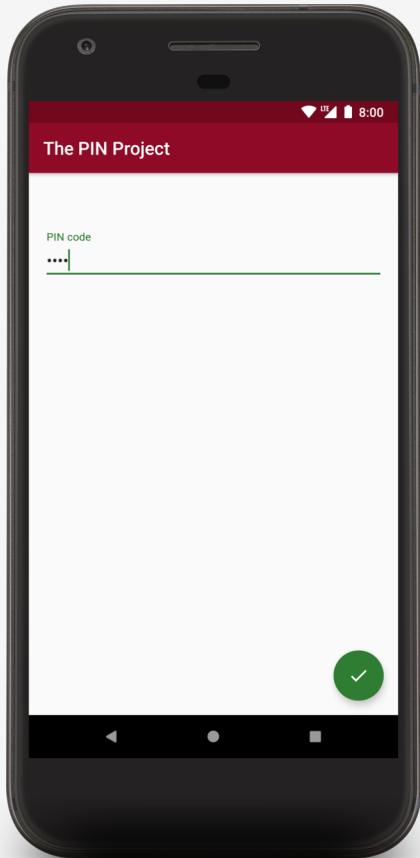


<https://github.com/RolandMostoha/the-pin-project/tree/step-3-Server-side-validation>



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Vizsgáljuk meg a tesztjeinket az új alkalmazással!

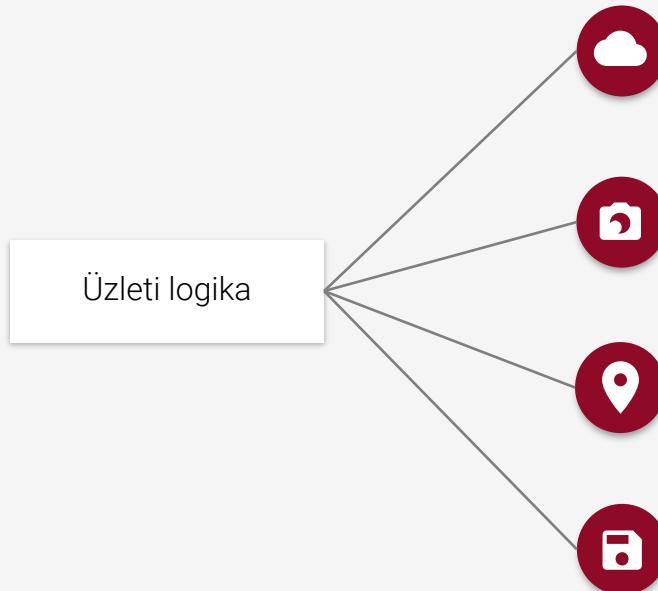
- Probléma: bejött egy külső függőség, ami eltöri a teszteket
- Nem determinisztikus a tesztelési környezet
- Nincs lehetőségünk tesztelni
 - A sikeres PIN-t követő navigációt
 - A sikertelen PIN-t jelző üzenetet

Megoldás: Hermetikus tesztelés



Tesztelési lehetőségek, Unit és UI tesztek

Hermetikus tesztelés



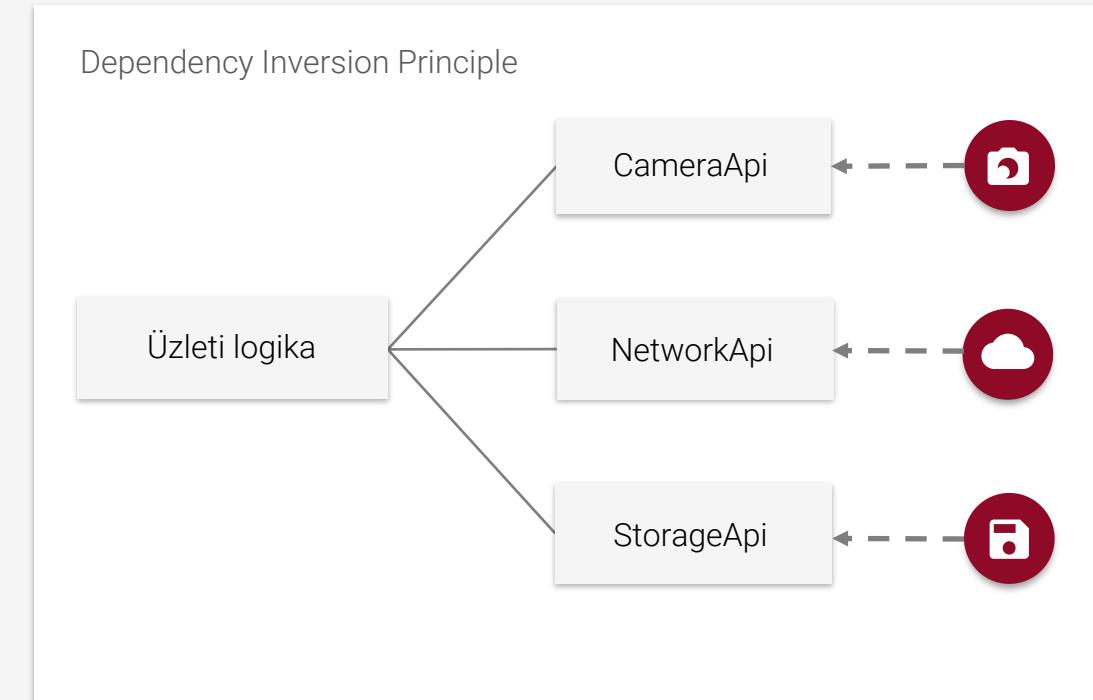
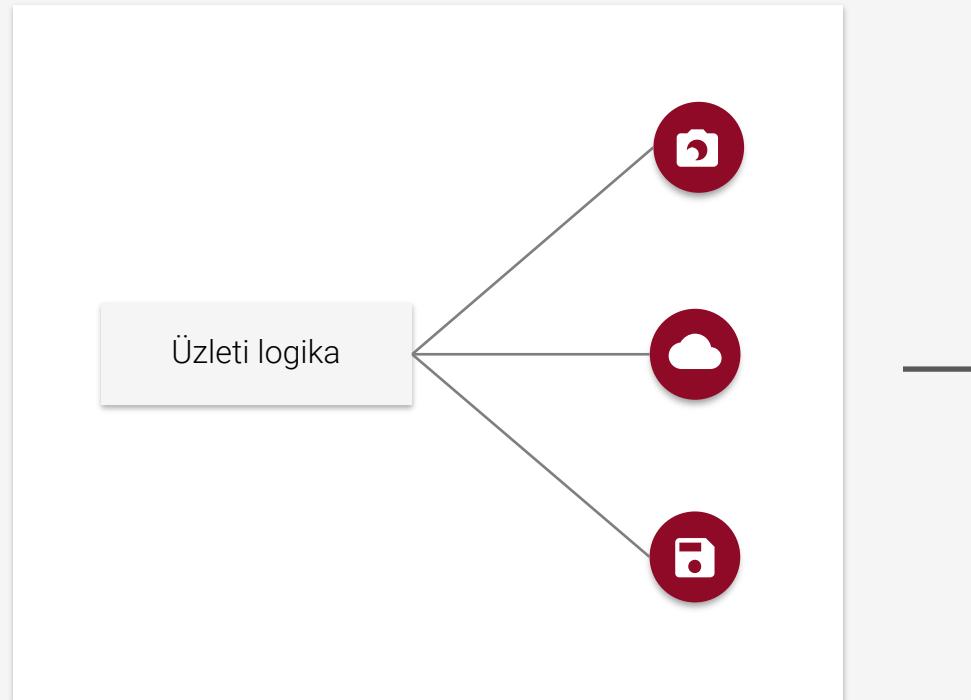
- Törékeny vagy "flaky" test: néha elbukik, néha sikeres
- Külső függőségek
 - Hibapontokat adnak a rendszerhez
 - Nagymértékben lassíthatják a tesztjeinket

Megoldás: A videlkedés leválasztása a külső függőségekről



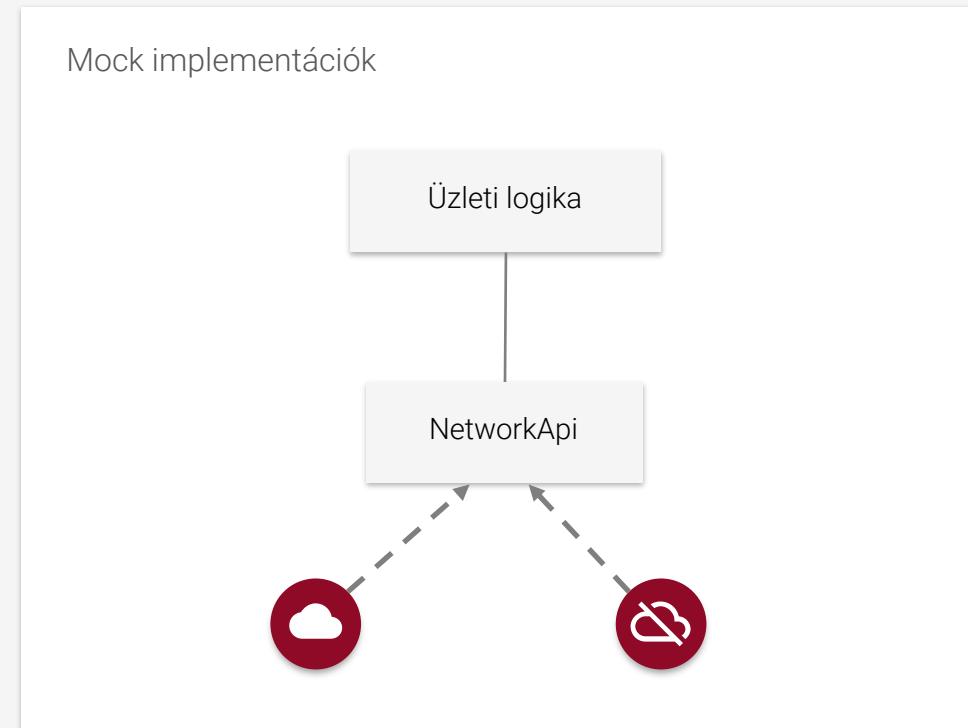
Mockolás

Dependency Inversion Principle



Dependency Inversion Principle

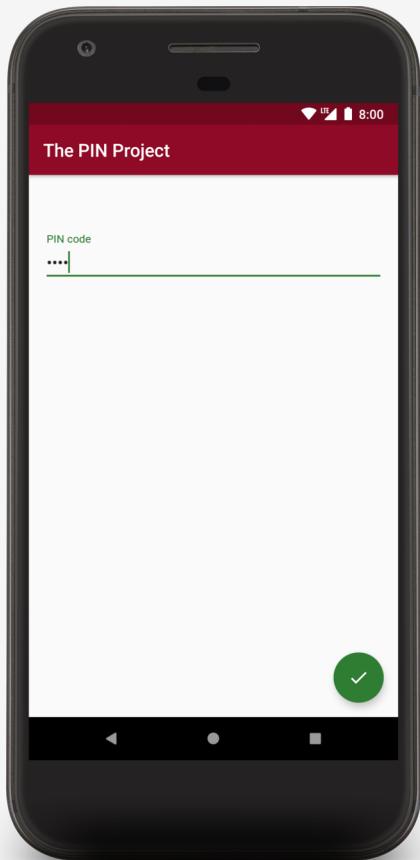
- Laza csatolás





Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Készítsünk egy Mock implementációt a validátorról!

- Késleltetés nélkül térjen vissza **TRUE** értékkel
- Probléma: hogyan mondjuk meg, hogy teszt környezetben a Mock implementációt használja?

Megoldás: Dependency Injection



Dependency Injection

- Egy osztály kiajánlja a függőségeit, a hívó félre bízza a példányosítást
 - Kevesebb a felelőssége, erősebb a kohéziója
- A következő problémákat hivatott megoldani:
 - Hogyan tud az osztályunk független maradni a függőségeinek példányosításától?
 - Hogyan tudjuk konfiguráció szinten változtatni ezeket a külső függőségeket?
 - Hogyan támogassunk különböző konfigurációkat?
- Több módszer létezik
 - Metódus injektálás – tipikusan Application, Activity, Fragment osztályokban, ahol nem mi végezzük a példányosítást
 - Konstruktur injektálás – mindenhol máshol



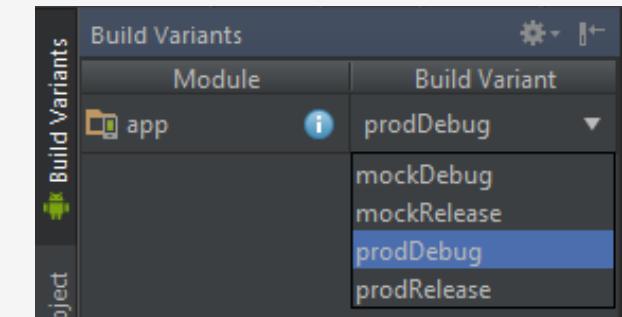
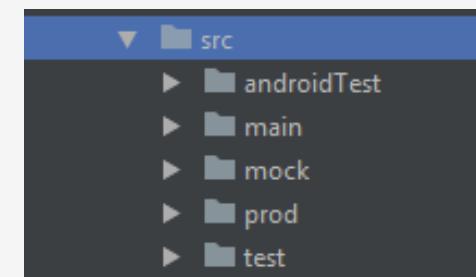
Dependency Injection keretrendszerek

- DI keretrendszerek léteznek, amelyek megkönnyítik a megvalósítását
 - Kiszervezik a példányosítást, automatikus injektálás
 - Pl. [Google Dagger 2](#)
- Extra funkciókat is tudnak
 - Pl. Singleton egy annotációval
 - Metódus injektálás @Inject annotációval
- Mind-mind ugyanarra az egyszerű elvre épít: osztályok viselkedése legyen független a függőségeinek példányosításától

Csináld magad DI framework Androidon

Mock és Prod konfigurációk készítése

```
productFlavors {  
    mock {  
        applicationIdSuffix = ".mock"  
    }  
    prod {  
    }  
}
```



Valós függőképek példányosítása

```
class Injection {  
    companion object {  
        fun provideValidatorApi(): PinValidatorApi {  
            return NetworkPinValidator()  
        }  
    }  
}
```

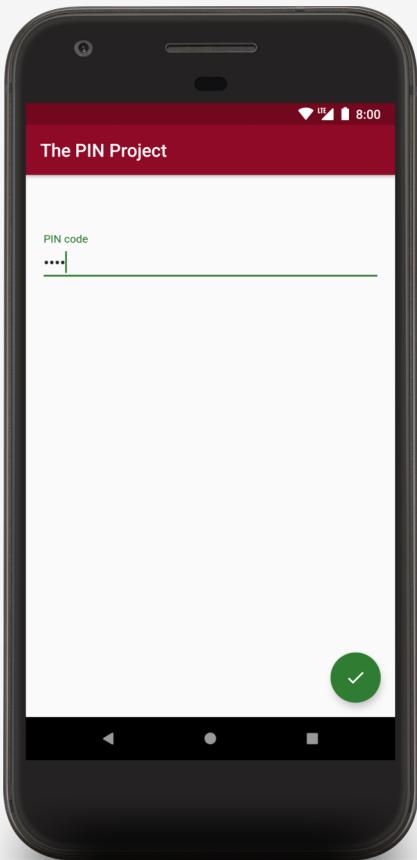
Mock függőképek példányosítása

```
class Injection {  
    companion object {  
        fun provideValidatorApi(): PinValidatorApi {  
            return MockPinValidator()  
        }  
    }  
}
```



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



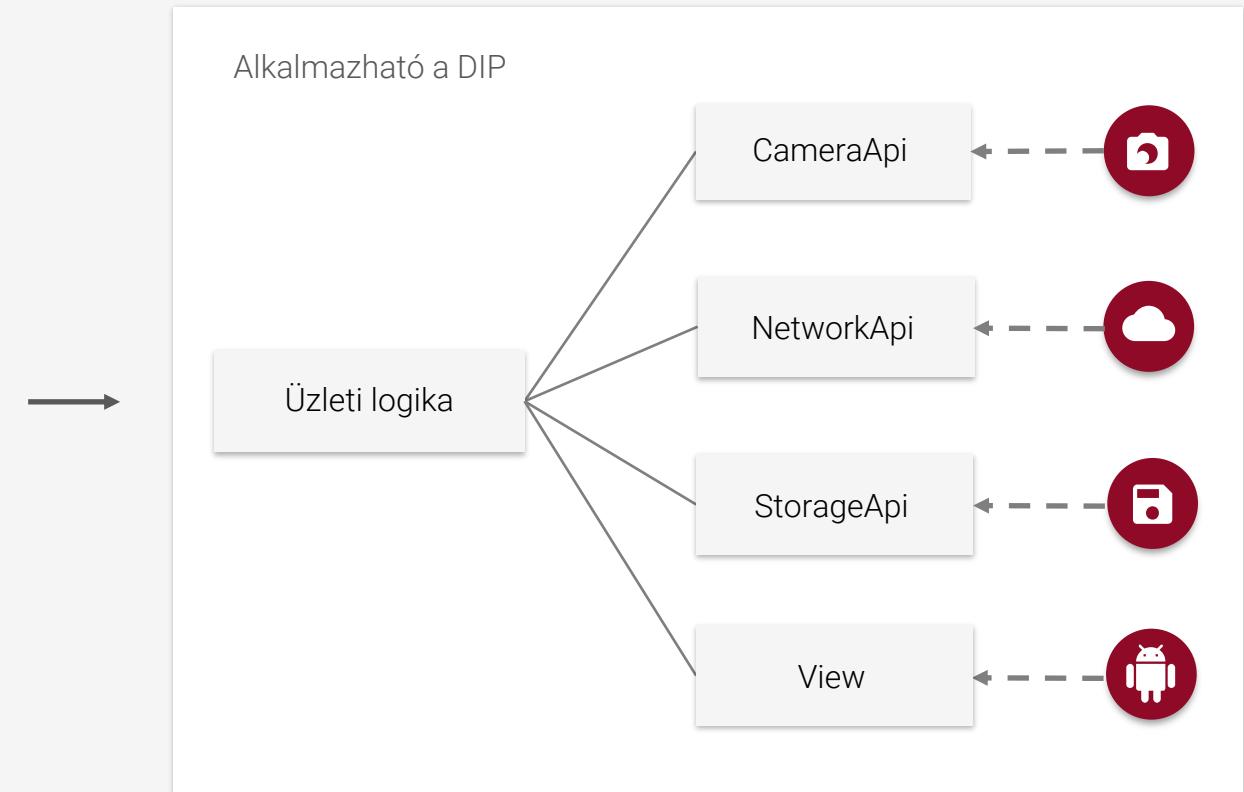
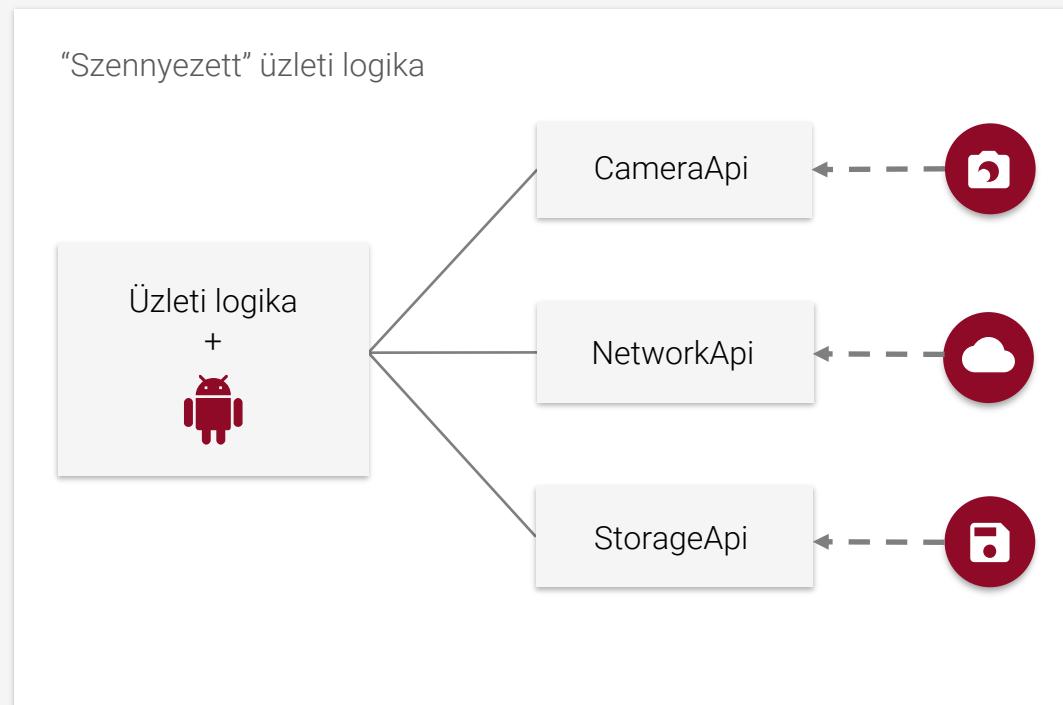
- ✓ Leválasztottuk a külső függőségünket (hálózat)
- ✓ Determinisztikus tesztelési környezetet alakítottunk ki
- ✓ Javítottunk a tesztjeink törékenységén
- ✓ Megalkottuk a saját kis DI keretrendszerünket ☺
- ✓ Architektúra laza csatolással, erős kohézióval



<https://github.com/RolandMostoha/the-pin-project/tree/step-4-DIP-DI>



Üzleti logika leválasztása



Architektúra az üzleti logika leválasztására

- Model-View-Presenter
 - Válasszuk le az Activity viselkedését egy interfésszel
 - Az Activity életciklus kezdetén csatoljuk fel, a végén pedig töröljük
 - Ezzel elkerüljük a hivatkozást a kontextusra
- Model-View-ViewModel
 - Google által javasolt architektúra, szinte ugyanazt az elvet követi
 - Extra funkciók
 - Felcsatolást, lecsatolást automatikusan végzi
 - Az Activity elforgatás után is ugyanazt a példányt adja

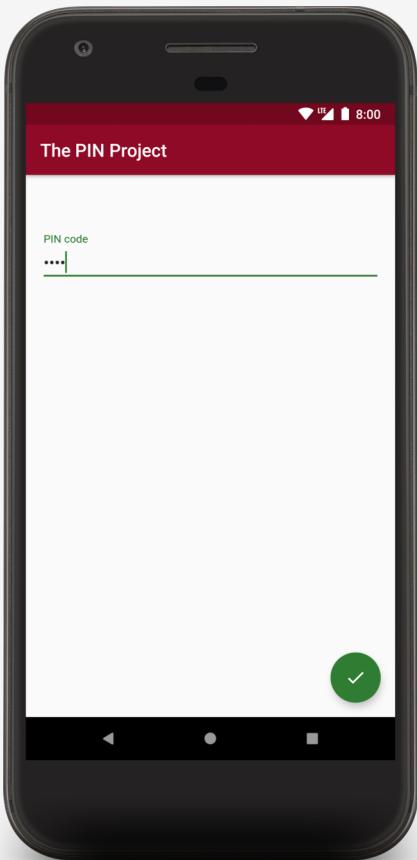


Caution: A ViewModel must never reference a view, Lifecycle, or any class that may hold a reference to the activity context.



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



Válasszuk le az Activity viselkedését a Controller osztályról

- PinView – Interfész az Activity viselkedésére
- PinPresenter – Az üzleti logikát tartalmazza
- Activity – Kizárolag View-val kapcsolatos műveleteket végez

Mockito



- Mock implementációkat gyárthatunk interfészektől és osztályokból
- Megadhatjuk, hogy bizonyos függvényekre mivel térjenek vissza
- Viselkedést ellenőrizhetünk az alapján, hogy bizonyos függvény meghívodott-e

```
@Test
fun `Given valid pin, when we validate it, navigateToMainScreen should be called`() {
    val pin = "7878"

    pinPresenter.validatePin(pin)

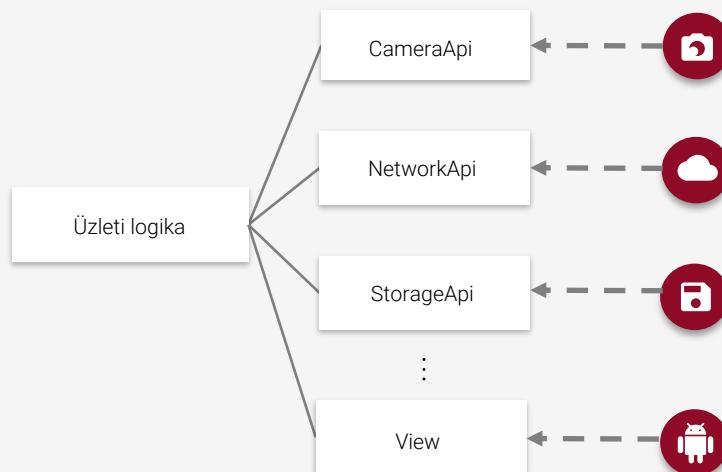
    verify(pinView).navigateToMainScreen()
}
```





Tesztelési lehetőségek, Unit és UI tesztek

Tesztelhető architektúra



- ✓ Az üzleti logika teljesen tiszta a Unit teszteléshez
- ✓ Karbantartható, minőségi kódot kaptunk
- ✓ Laza csatolás és erős kohézió jellemzi

Hogyan jutottunk el idáig? Mi csupán tesztelni szerettük volna az alkalmazásunkat.



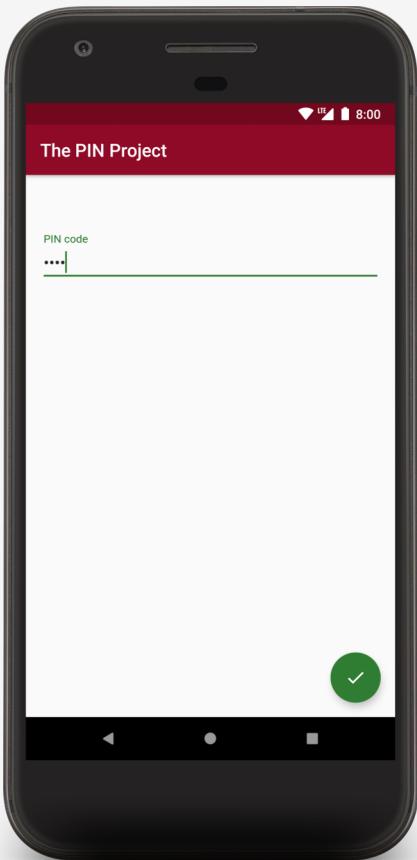
“

A jó architektúra nem követeli meg a tesztelést, de a tesztelés megköveteli a jó architektúrát



Tesztelési lehetőségek, Unit és UI tesztek

The PIN Project



- ✓ Clean Code
- ✓ Karbantartható, kiterjeszthető kód
- ✓ Kockázat-mentes refactor és funkcióbővítés
- ✓ Teljesen lefedtük Unit és UI tesztekkel az alkalmazást
- ✓ Minőségi, megbízható szoftver



<https://github.com/RolandMostoha/the-pin-project/tree/step-5-MVP>

Köszönöm a figyelmet!

mostoha.roland@autsoft.hu

@RolandMostoha