WYKŁAD 3

Projektowanie i planowanie aplikacji mobilnych

Od pomysłu do wdrożenia. Proces, zarządzanie zespołem i obowiązki lidera projektu.

2026 | Informatyka | dr inż. Mateusz Pomianek

- Faza ideacji i badania rynku
- Skład i role zespołu
- Agile / Scrum / Kanban
- UX/UI i roadmapa produktu
- Release & App Store Launch
- Metryki i ciągłe doskonalenie

Plan wykładu

1. **Kontekst rynkowy**
   Dlaczego planowanie jest kluczowe? Statystyki porażek
2. **Faza odkrywania**
   Ideacja, badania rynku, walidacja pomysłu
3. **Skład zespołu**
   Role, odpowiedzialności, modele współpracy
4. **Metodyki pracy**
   Scrum, Kanban, SAFe – wybór i konfiguracja
5. **Planowanie techniczne**
   Architektura, tech stack, roadmapa produktu
6. **Proces UX/UI**
   Design sprint, prototypowanie, testy użyteczności
7. **Development**
   Sprinty, code review, CI/CD, zarządzanie długiem technicznym
8. **Lider zespołu**
   Obowiązki, kompetencje, techniki zarządzania
9. **QA i testowanie**
   Strategie, automatyzacja, definicja Done
10. **Launch i po launchu**
    App Store, metryki, iteracja, zarządzanie kryzysem

ROZDZIAŁ 1

Kontekst rynkowy

Dlaczego planowanie jest kluczowe? Statystyki porażek i sukcesów

Dlaczego projekty mobilne zawodzą?

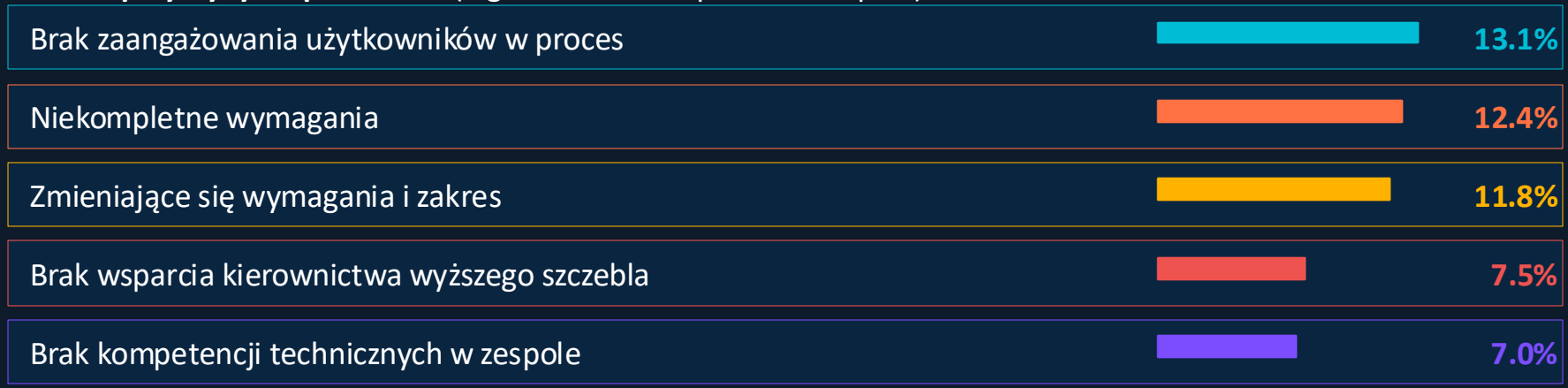
42%
projektów mobilnych
upada z powodu braku potrzeby rynkowej

68%
projektów IT
przekracza budżet lub harmonogram

77%
użytkowników
porzuca aplikację w ciągu pierwszych 3 dni

0.5%
aplikacji
osiąga finansowy sukces (Gartner, 2024)

Główne przyczyny niepowodzeń (wg Standish Group CHAOS Report)



Średni koszt nieudanego projektu mobilnego dla MŚP wynosi $240 000 (Clutch, 2024). Prawidłowe planowanie redukuje ryzyko o 35%.

ROZDZIAŁ 2

Faza odkrywania

Ideacja, badania rynku i walidacja pomysłu przed napisaniem pierwszej linii kodu.

Ideacja i walidacja pomysłu -> lean startup approach

1 **Znalezienie i rozpoznanie problemu**
Zdefiniuj problem, nie rozwiązanie. Użyj formatu: Dla [persona], która ma [problem], nasza aplikacja [rozwiązanie] daje [korzyść] w odróżnieniu od [alternatywy].

2 **Badania rynku**
TAM / SAM / SOM. Analiza konkurencji (App Annie, SensorTower).
Minimum 20 wywiadów z potencjalnymi użytkownikami przed projektem.

3 **Walidacja hipotez**
Build-Measure-Learn. Landing page z CTA + Google Ads = test gotowości rynku za ~500 zł.
Cel: >5% konwersja.

4 **Business Model Canvas**
9 bloków: segmenty, propozycja wartości, kanały, relacje, przychody, zasoby, aktywności, partnerzy, koszty.

Firmy inwestujące w fazę Discovery skracają czas developmentu o **33%** i redukują przeróbki o **72%** (Forrester, 2023). **Każda złotówka** wydana na badania **oszczędza 10 zł** na poprawkach.

Analiza konkurencji i definicja MVP

Analiza konkurencji

Kryterium	Twoja app	Konkurent A	Konkurent B
Czas do wartości (TTFV)	< 60s	3 min	5 min
Monetyzacja	Freemium	Paid	Ads
Offline support	Tak	Nie	Częściowy
Ocena App Store	–	3.8★	4.1★

Definicja MVP: Minimum Viable Product

Must Have (MoSCoW)
Funkcje bez których produkt nie ma sensu. Maks. 3-5 dla pierwszego releasu.

Should Have
Ważne, ale można wydać bez. Dodawane w sprint 2-3.

Could Have
Nice-to-have. Backlog priorytet B.

Won't Have (v1)
Explicite wykluczone z releasu 1.0. Zarządzanie oczekiwaniami.

Spotify wypuścił MVP z jedną funkcją (streaming muzyki), bez playlist, offline, czy social.
Instagram startował wyłącznie jako apka do filtrowania zdjęć (bez stories, reels, zakupów).

ROZDZIAŁ 3

Skład i role zespołu

Kto jest potrzebny i za co odpowiada. Modele współpracy i struktury organizacyjne.

Role w zespole mobilnym: odpowiedzialności i kompetencje

Product Owner (PO)

Właściciel backlogu. Priorytety feature'ów. Jedyna osoba która może powiedzieć NIE. Odpowiada za ROI produktu. Czas: 100% projektu.

Scrum Master / Agile Coach

Facylituje ceremonie. Usuwa przeszkody (impediments). NIE zarządza zespołem („servant leader").

Tech Lead / Lider Techniczny

Decyzje architektoniczne. Code review. Mentoring juniora. 30% zarządzania, 70% kodowania.

iOS / Android Developer

Implementacja feature'ów. Unit testy. Uczestnictwo w code review. Senior: 5+ lat. Mid: 2-5 lat. Junior: 0-2 lata.

UX/UI Designer

Research, wireframes, prototypy w Figma. Design system. Pracuje sprint do przodu. Niezbędny od dnia 1.

QA Engineer

Plan testów, przypadki testowe, automatyzacja (Appium, Detox). Definicja Done. Bug triage.

Backend Developer

API REST/GraphQL, baza danych, integracje. Mobile-first API design. Na wczesnym etapie = fullstack.

DevOps / Platform Eng.

CI/CD pipeline (GitHub Actions/Fastlane), EAS Build, monitoring, App Store deployment automation.

Modele pracy zespołu: insourcing, outsourcing, nearshoring

Inhouse (własny zespół)

Zalety

- ✓ Pełna kontrola kultury
- ✓ Głęboka znajomość produktu
- ✓ Szybka komunikacja
- ✓ IP w firmie

Wady

- ✗ Wysokie koszty rekrutacji
- ✗ Czas onboardingu 2-6 mies.
- ✗ Trudno skalować szybko
- ✗ Ryzyko rotacji

~$8 000–25 000/mies. per dev (PL)

Software House (outsourcing)

Zalety

- ✓ Szybki start (2-4 tyg.)
- ✓ Kompletny zespół od razu
- ✓ Znany koszt miesięczny
- ✓ Elastyczne skalowanie

Wady

- ✗ Niższa znajomość domeny
- ✗ Rotacja zasobów
- ✗ Komunikacja overhead
- ✗ Mniej zaangażowania

~$4 500–12 000/mies. per dev (PL)

Model hybrydowy

Zalety

- ✓ Inhouse core team
- ✓ Outsourcing pik'ów
- ✓ Optymalne koszty
- ✓ Transfer wiedzy

Wady

- ✗ Złożone zarządzanie
- ✗ Dwa procesy
- ✗ Ryzyko silosów
- ✗ Wymaga dojrzałości

Używany przez 58% scale-up'ów (Deloitte)

„Two-pizza rule” - jeśli zespół nie może być nakarmiony dwoma pizzami, to jest za duży.
Optymalny rozmiar interdyscyplinarnego zespołu to około 5-9 osób (Scrum Guide).

ROZDZIAŁ 4

Metodyki pracy

Scrum, Kanban, SAFe: wybór i konfiguracja dla projektów mobilnych.

Scrum: ceremonie i artefakty

Sprint Planning — **2h / 2 tyg.**

PO + Dev Team. Cel sprintu + selekcja backlogu. Velocity = punkty story z poprzednich 3 sprintów.

Daily Standup — **15 min / dzień**

Co robiłem? Co robię? Co blokuje? Stojąc. Bez rozwiązywania problemów on-the-spot. Parking lot.

Sprint Review — **1h / 2 tyg.**

Demo dla stakeholders. Tylko działający software. Zbieranie feedbacku. Aktualizacja backlogu.

Retrospective — **1.5h / 2 tyg.**

Stop / Start / Continue. 4Ls (Liked/Learned/Lacked/Longed For). Action items z właścicielem.

Backlog Refinement — **1h / tyg.**

PO + Tech Lead. Story pointing (Fibonacci). Kryterium akceptacji (Given/When/Then). Dekompozycja.

Dynamika zespołu (5 dev): ~40-80 story points/sprint. Sprint 1 zawsze wolniejszy (setup CI/CD, architektura). Mierz, nie estymuj godzinami (**prawo Hofstadtera**)

Kanban, SAFe i porównanie metodyk: kiedy co stosować

Kanban board: przykład dla mobile

Backlog · WIP: ∞	In Progress · WIP: 2	Review · WIP: 1	Done · WIP: ∞
Push notifications	Auth flow	Login screen UI	Onboarding
Dark mode	API cache layer		Home screen
iPad layout			

Porównanie metodyk

Kryterium	Scrum	Kanban	SAFe
Rozmiar zespołu	5–9 osób	1–50+ osób	50–5000 osób
Cykl releasu	Co sprint (2 tyg.)	Ciągły flow	Program Increment (PI)
Zmiany w trakcie	Tylko nowy sprint	W każdej chwili	Planowane kwartalnie
Narzut administracyjny	Średni	Niski	Wysoki

Rekomendacja: Startup/MVP → Kanban (prostota). **Zespół** 5-15 osób → Scrum. **Kilka zespołów mobilnych** → SAFe lub LeSS (Large-Scale Scrum).

ROZDZIAŁ 5

Planowanie techniczne

Architektura, tech stack i roadmapa produktu.

Architecture Decision Records (ADR)

Po co to komu?

Kontekst — Dlaczego decyzja jest potrzebna? Co nas do tej decyzji zmusiło?

Decyzja — Co zdecydowaliśmy? Jedno zdanie aktywne: "Używamy X ponieważ Y".

Status — Proposed / Accepted / Deprecated / Superseded

Konsekwencje — Pozytywne (zalety) i negatywne (kompromisy). Jawne trade-offs.

Kluczowe decyzje architektoniczne

Native vs Cross-Platform

React Native / Flutter jeśli team<8 i budżet<2M PLN. Native jeśli performance-critical lub AR/VR.

State Management

MobX (prosta apka), Redux/RTK (complex). iOS: Combine+SwiftUI. Android: StateFlow+ViewModel.

Offline first?

SQLite / Room / Realm + sync strategy. Definiuj conflict resolution PRZED coding.

API contract

OpenAPI 3.0 spec PRZED implementacją. Contract testing (Pact). Consumer-driven.

Badania ThoughtWorks: zespoły które dokumentują ADR mają o 40% mniej nieplanowanych spotkań architektonicznych. ADR przechowuj w repozytorium (katalog **/docs/adr**).

Architecture Decision Records (ADR)

ADR-001: Wybór architektury aplikacji Android

Status
Accepted

Data
2026-05-26

Kontekst
Projektujemy aplikację mobilną na Androida, która ma obsługiwać logowanie użytkownika, pracę z API, lokalne przechowywanie części danych oraz dalszą rozbudowę o moduły powiadomień i analityki.

Zespół jest niewielki, dlatego architektura powinna być czytelna, łatwa do testowania i możliwa do utrzymania przez nowych programistów.

Decyzja
Przyjmujemy architekturę MVVM z podziałem na warstwy:

- **UI**: Jetpack Compose
- **ViewModel**: obsługa stanu ekranu i logiki prezentacji
- **Domain**: przypadki użycia aplikacji
- **Data**: repozytoria, API, baza lokalna
- **Dependency** Injection: Hilt
- **Local storage**: Room
- **Networking**: Retrofit + OkHttp
- **Async**: Kotlin Coroutines + Flow

Uzasadnienie
MVVM dobrze pasuje do aplikacji Android tworzonych w Kotlinie i Jetpack Compose. Pozwala oddzielić logikę interfejsu od logiki biznesowej, ułatwia testowanie oraz zmniejsza ryzyko mieszania kodu UI z obsługą danych.

Jetpack Compose upraszcza budowanie interfejsu, a Flow pozwala wygodnie reagować na zmiany stanu aplikacji.

Alternatywy
1. **MVC**
   Odrzucone, ponieważ prowadzi do silnego powiązania widoku z logiką aplikacji.

2. **MVP**
   Odrzucone, ponieważ zwiększa liczbę klas i jest mniej naturalne w połączeniu z Compose.

3. **Clean Architecture** w pełnej wersji
   Odrzucone dla MVP, ponieważ byłaby zbyt rozbudowana na obecny zakres projektu.

Konsekwencje pozytywne
- lepsza testowalność kodu,
- czytelny podział odpowiedzialności,
- łatwiejszy onboarding nowych osób,
- mniejsze ryzyko długu technicznego,
- dobra integracja z ekosystemem Android Jetpack.

Konsekwencje negatywne
- większa liczba plików i warstw,
- konieczność zachowania dyscypliny architektonicznej,
- początkowo wolniejsze tempo pracy dla mniej doświadczonych programistów,
- ryzyko nadmiernej abstrakcji przy prostych funkcjach.

Kryterium rewizji decyzji
Decyzja powinna zostać ponownie oceniona, jeśli:
- aplikacja znacząco urośnie funkcjonalnie,
- pojawi się potrzeba wersji iOS,
- logika domenowa stanie się złożona,
- testowanie obecnej struktury okaże się niewystarczające,
- zespół zacznie mieć trudności z utrzymaniem spójności kodu.

Roadmapa produktu

Faza 0 **Discovery**

Mies. 1-2

- Research & walidacja
- Arch. decisions
- Design sprint
- Team onboarding

Faza 1 **MVP**

Mies. 3-5

- Core features
- CI/CD setup
- Alpha testing
- Soft launch

Faza 2 **Growth**

Mies. 6-8

- User feedback loop
- Performance opt.
- A/B testy
- Push / retention

Faza 3 **Scale**

Mies. 9-12

- Monetyzacja
- New platforms
- Refactor debt
- Team scale-up

Struktura kosztów projektu (typical mid-size app, rynek PL)

Development (iOS+Android)	45–55%
UX/UI Design	12–18%
Backend / API	10–15%
QA & Testing	8–12%
PM & Discovery	5–8%
DevOps / Infra	3–5%

Roadmapa dla aplikacji mobilnej na Androida

```
# M1: Discovery + UX
- walidacja problemu
- MVP, prototyp
- testy z użytkownikami

# M2: Architektura + MVP
- ADR
- MVVM
- API
- baza lokalna
- podstawowe funkcje

# M3: Testy + soft launch
- QA, monitoring
- beta testy, Google Play closed testing
- poprawki

# M4: Public launch
- Publikacja
- staged rollout
- analiza crashy
- retencji i opinii

# M5–M6: Growth
- push notifications
- optymalizacja UX
- A/B testy
- Monetyzacja
- skalowanie
```

```
# Roadmapa aplikacji Android – MVP → Launch

## Milestone 1: Walidacja pomysłu
Termin: koniec 2. tygodnia
Rezultat: potwierdzony problem użytkownika, opis person, zakres MVP.

## Milestone 2: Prototyp UX/UI
Termin: koniec 4. tygodnia
Rezultat: klikalny prototyp, przetestowane główne ścieżki użytkownika.

## Milestone 3: Architektura techniczna
Termin: koniec 5. tygodnia
Rezultat: ADR, struktura projektu, wybrany stack technologiczny.

## Milestone 4: MVP
Termin: koniec 10. tygodnia
Rezultat: działająca aplikacja z podstawowymi funkcjami.

## Milestone 5: Beta
Termin: koniec 13. tygodnia
Rezultat: aplikacja przetestowana na ograniczonej grupie użytkowników.

## Milestone 6: Launch
Termin: koniec 14. tygodnia
Rezultat: publikacja w Google Play.

## Milestone 7: Wersja 1.1
Termin: koniec 18. tygodnia
Rezultat: poprawki UX, stabilizacja, pierwsze funkcje po feedbacku.
```

ROZDZIAŁ 6

Proces UX/UI Design

Design sprint, prototypowanie i testy użyteczności.

Design Sprint i UX Research: od problemu do przetestowanego prototypu

Pon. **Understand**	Wt. **Sketch**	Śr. **Decide**	Czw. **Prototype**	Pt. **Test**
Map problemu, wywiady eksperckie, HMW (How Might We), cel długoterminowy.	Crazy 8's, Lightning demos, Solution sketch. Każdy osobno.	Dot voting, Decider vote, storyboard (15 kroków na flipcharcie).	Figma high-fidelity. Tylko to co zobaczą testerzy, a nie cały flow.	5 użytkowników (Jakob Nielsen). Moderowane wywiady. Think-aloud protocol.

Kluczowe metryki UX i kiedy badać

Task Completion Rate — **>85%**
Kiedy: Po wireframe i po launch · <70% = poważny problem nawigacyjny

Time on Task — **Benchmark**
Prototyp vs produkcja · 3× dłużej niż ekspert = zły UX

System Usability Scale (SUS) — **>68 pkt**
Beta testing · >85 = excellent. Apple produkty: ~80

App Store Rating — **>4.2★**
Po launch i per release · <3.8★ = problem retencyjny

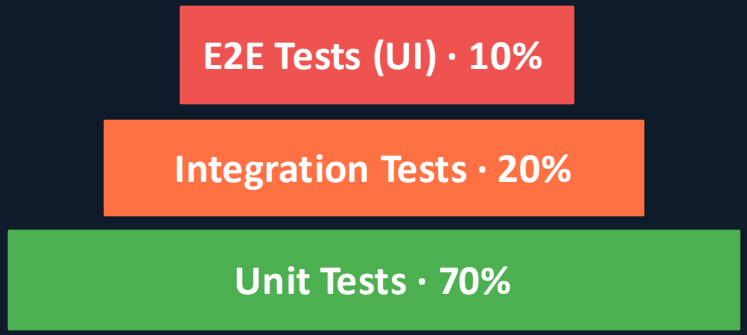
ROZDZIAŁ 9

QA i strategia testowania

Piramida testów, automatyzacja, definicja Done i zarządzanie długiem technicznym.

Piramida testów mobilnych: strategia i automatyzacja

Piramida testów (Mike Cohn)



Detox, Maestro, Appium
Najdrozsze, wolne, kruche

XCTest, Espresso, MSTest
Srednie. API + DB + UI integracja

Jest, JUnit, XCTest, Mockito
Szybkie <1s, tanie, deterministyczne

Definicja Done (DoD): wzorcowa dla mobile

Kod — Code review przez ≥1 seniora. Brak komentarzy TODO. Linting = 0 errors.

Testy — Unit test coverage ≥80%. Regression suite przeszedł. Brak P0/P1 bugów.

Wydajność — < 2s cold start. < 60MB memory footprint. Frame rate ≥ 60fps na average device.

Dostępność — WCAG 2.1 AA. TalkBack/VoiceOver testowany. Contrast ratio ≥4.5:1.

Dokumentacja — README zaktualizowane. API changelog. Release notes draft.

Code review redukuje liczbę bugów o 60% (SmartBear, 2023).
Każdy błąd znaleziony w fazie testów, będzie kosztował 10× więcej do naprawienia po wypuszczeniu apki.

Zarządzanie długiem technicznym: klasyfikacja i strategie

23%
czasu dev
spędzone na tech debt
(McKinsey 2023)

40%
wolniejszy
development przy wysokim tech debt

$1.52T
globalny koszt
długu technicznego
(CISQ 2023)

20%
sprint
na spłatę długu
(rekomendacja)

Rodzaje długu technicznego i strategie spłaty

Deliberate (świadomy)

"Zrobimy to szybko teraz, naprawimy później." Akceptowalny jeśli jest decyzja świadoma.

Action: Ticket w backlogu z datą spłaty. Nie starszy niż 2 sprinty.

Inadvertent (nieświadomy)

"Nie wiedzieliśmy że tak nie powinno się robić." Pojawia się przy wzroście know-how lub zmianie standardów.

Action: Coding guidelines + regular Tech Debt Sprint co Q.

Rotten Code (degradacja)

Biblioteka nieaktualizowana od 2 lat. Deprecated API. Kod który "nikt nie dotyka bo boi się". Timebomb.

Action: Dependency audit co sprint. Automatyczne Dependabot/Renovate.

Test Debt

Brak testów, flaky testy ignorowane, manual-only regression. Każdy deploy to roulette.

Action: Coverage gate w CI (SonarQube). Brak merge jeśli coverage spada.

ROZDZIAŁ 10

Launch, launch i po-launchu

App Store submission, metryki sukcesu, zarządzanie kryzysem i iteracja.

Checklista przed publikacją

- ☐ **App Store Screenshots**
  6.7" + 6.5" + 12.9" + 5.5". Tekst marketingowy na grafice.
- ☐ **App Preview Video**
  15-30s. Pierwsze 3 sekundy krytyczne. Bez lektora/głosu.
- ☐ **Keywords (ASO)**
  Max 100 znaków. Nie powtarzaj nazwy. Używaj SensorTower.
- ☐ **Privacy Policy URL**
  HTTPS. Wymagana dla apps z logowaniem lub analityką.
- ☐ **Crash rate < 1%**
  TestFlight beta ≥ 500 users, 7 dni bez P0 bugów.
- ☐ **Accessibility audit**
  VoiceOver / TalkBack. Dynamic Type. High Contrast.
- ☐ **Performance baseline**
  < 3s cold start. < 200ms API latency p95.
- ☐ **Staged rollout plan**
  1% → 5% → 20% → 100% w 3 tygodniach (Android).
- ☐ **Rollback procedure**
  Przygotuj poprzedni build. Max 1h do rollback decision.
- ☐ **Monitoring & alerts**
  Crashlytics, Firebase Perf, custom alerts w PagerDuty.

ASO Stat: 65% pobrań zaczyna się od wyszukiwania w App Store (Apple).
Aplikacje z wideo preview konwertują o 35% lepiej. Ocena 4.0→4.5★ = +89% więcej pobrań.

Metryki sukcesu

A	**Acquisition** CPI (Cost per Install)	Target: iOS: <$2, Android: <$0.8	App Store Organic + Paid UA (Meta/TikTok Ads). Źródła: Search, Browse, Referral
A	**Activation** Day-1 Retention	Target: >40%	Czy użytkownik wrócił dzień po instalacji? Onboarding flow kluczowy.
R	**Retention** Day-7 / Day-30	Target: D7>20%, D30>10%	Industry avg: D7=13%, D30=6%. Top apps: D7=50%+. Cohort analysis w Amplitude.
R	**Revenue** ARPU / LTV	Target: LTV > 3× CPI	ARPU = MRR / MAU. LTV = ARPU × średni czas życia użytkownika.
R	**Referral** Viral Coefficient (K)	Target: K > 0.5	K = Invites × Conversion rate. K>1 = viral growth (rzadkie). NPS > 50 = excellent.

North Star Metric = jedna liczba która wyraża wartość produktu dla użytkownika.
Spotify: minuty odsłuchu. // **Airbnb**: zarezerwowane noce. // **WhatsApp**: wiadomości wysłane/dzień.

Zarządzanie kryzysem produkcyjnym: Incident Management

$5 600
utrata przychodu
na minutę downtime (avg e-commerce)

3★
porzucenie apki
użytkownicy odinstalowują przy ocenie <3★

72h
okno krytyczne
wizerunek można uratować w 72h od incydentu

Klasyfikacja incydentów i czas reakcji

P0 — Critical
Apka nie startuje / crash rate >10% / utrata danych użytkowników
Reakcja: **15 min** Rozwiązanie: 2h

P1 — High
Kluczowy flow nie działa (logowanie, płatności) / degradacja >50% userów
Reakcja: **1h** Rozwiązanie: 8h

P2 — Medium
Feature działa z błędami / błąd dla <10% userów / brak danych statystycznych
Reakcja: **4h** Rozwiązanie: 2 dni

P3 — Low
Drobny błąd UI / literówka / feature nieużywany
Reakcja: **24h** Rozwiązanie: sprint

Post-mortem
Co się stało? → Dlaczego? (5×Why) → Co robimy żeby się nie powtórzyło? → Action items
Kultura blameless jest konieczna, aby ludzie raportowali incydenty wcześniej.

Iteracja i ciągłe doskonalenie: OKR, feedback loops i roadmapa v2

CEL: Stać się wiodącą aplikacją fitness w Polsce do Q4 2026

KR1	MAU wzrośnie z 50k do 200k do 31.12.2026	Amplitude DAU/MAU ratio
KR2	Day-30 retention wzrośnie z 12% do 25%	Cohort analysis w Mixpanel
KR3	App Store rating wzrośnie do 4.5★ (min. 5000 reviews)	App Store Connect
KR4	MRR z subskrypcji osiągnie 150 000 PLN	RevenueCat dashboard

Pętle feedbackowe: ciągła poprawa produktu

Codziennie

Źródło: Crash reports, App Store reviews, support tickets
→ Hotfix jeśli P0/P1, triaging reszty

Co sprint

Źródło: Sprint review demo, Analytics dashboard, A/B tests
→ Adjustment backlogu, re-prioritization

Miesięcznie

Źródło: NPS survey, user interviews (5 os.), churn analysis
→ Roadmap update, feature kill/keep/build

Kwartalnie

Źródło: OKR review, market benchmark, competitor analysis
→ Strategia produktu, tech investment plan

Amazon: "Every day is launch day", czyli ciągłe wydawanie małych zmian jest bezpieczniejsze, niż duży release co pół roku.
Netflix wypuszcza setki deploymentów dziennie.

ROZDZIAŁ 8

Lider zespołu

Obowiązki, kompetencje, techniki zarządzania i częste błędy

Techniki zarządzania zespołem i komunikacja

Struktura spotkania 1:1 (30 min/tydzień)

0–5 min	**Check-in**	Samopoczucie, nastrój, co poza pracą
5–15 min	**Postęp**	Co działa? Co nie? Gdzie potrzebujesz pomocy?
15–22 min	**Feedback**	Konkretny, rzeczowy. SBI: Situation-Behavior-Impact
22–28 min	**Rozwój**	Gdzie chcesz być za rok? Co zrobisz do następnego 1:1?
28–30 min	**Action items**	Max 2-3 konkretne działania z właścicielem i terminem

Macierz komunikacji projektu

Daily Standup
Cały team · Codziennie 9:00 · 15 min
Async-first (loom) gdy distributed

Sprint Review
Team + stakeholders · Koniec sprintu · 1h
Demo TYLKO działającego software

1:1
Lider ↔ Developer · Co tydzień · 30 min
Prywatne, poufne, focus na osobę

Risk Review
PO + Tech Lead · Co 2 tyg. · 30 min
Risk register update, escalation

All-hands
Firma · Miesięcznie · 1h
Vision, metrics, Q&A z CEO

Antywzorce lidera, czyli czego UNIKAĆ:

- ✗ Micromanagement: "dlaczego to zajęło 3h?" – niszczy autonomię i motywację
- ✗ Brak feedbacku: milczenie ≠ akceptacja. Pracownik domyśla się błędnie
- ✗ "Heroic programmer": lider sam ratuje sprinty, a to blokuje wzrost zespołu

Zasady zarządzania

1. Jasność jest ważniejsza, niż intensywność. Zespół nie potrzebuje ciągłej presji, tylko jasnych priorytetów.
2. Nie zarządzać przez domysły. Jeżeli coś jest ważne, trzeba to nazwać.
3. Organizować pracę w małych, kontrolowalnych porcjach. Dbać o rytm pracy, stały cykl planowania, jasne zadania.
4. Chronić zespół przed nadmiarem pracy i obserwować przeciążenie.
5. Feedback powinien być szybki, konkretny i spokojny.
6. Code review nie może być areną dominacji. Odróżniać kontrolę, od odpowiedzialności.
7. Spotkania zawsze muszą mieć funkcję.
8. Nie być centrum zespołu, tylko projektantem warunków dla jego działania.

Dojrzałe zarządzanie to **mniej improwizacji**, **mniej presji**, **mniej chaosu,**
a **więcej jasnych reguł**, **dobrego feedbacku** i **świadomej pracy**.

Obowiązki lidera

1 Wizja techniczna i decyzje archt.

Przygotowuje i moderuje ADR. Utrzymuje architectural fitness function. Organizuje Architecture Review Board (ARB) raz na sprint. Odpowiada za tech debt roadmap.

2 Zarządzanie ryzykiem projektu

Prowadzi Risk Register (prawdopodobieństwo × wpływ). Identyfikuje single points of failure w zespole. Eskaluje blokery do PM w ciągu 24h.

3 Mentoring i rozwój zespołu

1:1 minimum 30 min/tydzień per osoba. Individual Development Plan (IDP) dla każdego. Pair programming juniora z seniorem ≥20% czasu.

4 Jakość kodu i standardy

Ustala coding guidelines (EditorConfig, linting, Detekt/SwiftLint). Pull Request template. Code review: 80% coverage minimum. Definicja Done egzekwowana.

5 Komunikacja z inwestorami

Weekly status report (2 akapity: osiągnięcia + ryzyka). Demo per sprint. Tłumaczy complexity na język biznesowy. Buffer między tech a biznesem.

6 Planowanie zasobów i mocy przerobowych

Uwzględnia urlopy, L4, onboarding nowych. Buffor na unplanned work: 20% sprint velocity. Nie obiecuje 100% capacity na nowe features.

7 Kultura i dobrostan zespołu

Mierzy morale zespołu (monthly mood check). Celebruje sukcesy (retrospective appreciation round).
Zapobiega burnoutowi: max 45h/tydzień jako norma.

Podsumowanie

1 Waliduj zanim zakodujesz

Każdy tydzień walidacji oszczędza miesiąc developmentu. 42% projektów upada z powodu braku potrzeby rynkowej.

2 Planowanie to inwestycja

Discovery + ADR + roadmapa = mniej zaskoczeń. Brak planu = plan na niepowodzenie.

3 Ludzie ponad procesy

Najlepszy stack technologiczny nie zastąpi dobrego zespołu z jasną wizją i dobrym liderem.

4 Mierz wszystko

AARRR, OKR, DORA metrics (Deployment Frequency, Lead Time, MTTR, Change Failure Rate).

5 Iteruj szybko, myśl długo

Małe częste releasy, duże strategiczne decyzje rzadziej. Build-Measure-Learn loop.

6 Dług techniczny == hipoteka

Spłacaj regularnie 20% sprintu. Zaniedbany rośnie z odsetkami i spowalnia cały projekt.

*"**Plans are worthless, but planning is everything.**" - Dwight D. Eisenhower*

Programowanie aplikacji wymaga **tyle samo inżynierii organizacyjnej, co technicznej**

Pytania do wyładu 2

Które elementy najlepiej opisują różnicę pomiędzy UI, a UX?

A. UI obejmuje wyłącznie logikę biznesową aplikacji
B. UX dotyczy tylko warstwy wizualnej interfejsu
C. UI to warstwa wizualna i interaktywna
D. UX to całościowe doświadczenie użytkownika

Które implikacje wynikają z prawa Fittsa?

A. dokładność interakcji jest niezależna od wielkości celu
B. powinno się minimalizować różnice w rozmiarach elementów
C. większe elementy są łatwiejsze do trafienia
D. elementy powinny być blisko naturalnej pozycji palca

Które zasady dotyczą poprawnego użycia kolorów w UI?

A. stosowanie dużej ilości kolorów
B. brak potrzeby testowania w różnych warunkach oświetlenia
C. użycie kontrastu do kierowania uwagą użytkownika
D. uwzględnienie dostępności (np. daltonizm)

Które cechy są kluczowe dla projektowania mobilnego?

A. duża przestrzeń robocza ekranu
B. precyzja kursora myszy
C. ograniczona przestrzeń i konieczność priorytetyzacji treści
D. interakcje dotykowe i strefa kciuka

Które zasady są zgodne z podejściem Mobile First?

A. projektowanie pełnej funkcjonalności przed optymalizacją
B. traktowanie mobilnej jako uproszczonej wersji desktopowej
C. priorytetyzacja treści i funkcji kluczowych
D. progressive enhancement

Które cechy są charakterystyczne dla dobrej typografii mobilnej?

A. maksymalne zagęszczenie tekstu na ekranie
B. stosowanie wielu krojów pisma dla różnorodności
C. czytelność wsparta odpowiednią interlinią
D. hierarchia typograficzna wspierająca skanowanie treści

Które interfejsy zaliczamy do naturalnych form interakcji?

A. CLI
B. CUI
C. NUI
D. TUI

Które elementy należą do projektowania emocjonalnego?

A. optymalizacja algorytmów przetwarzania danych
B. minimalizacja liczby animacji do zera
C. poziom visceral związany z pierwszym wrażeniem
D. poziom behavioral związany z użytecznością

Które zasady dotyczą projektowania haptyki?

A. stosowanie haptyki jako podstawowego kanału komunikacji
B. kontekstu użycia aplikacji nie ma znaczenia
C. dostarczanie subtelnego feedbacku o stanie interfejsu
D. ograniczenie użycia do kluczowych interakcji

Które zasady wynikają z prawa Hicka?

A. zwiększanie liczby opcji przyspiesza decyzję
B. użytkownik preferuje maksymalną liczbę wyborów
C. czas decyzji rośnie wraz z liczbą opcji
D. ograniczanie wyborów poprawia UX

Które elementy są istotne w standardach WCAG?

A. estetyka interfejsu
B. dowolny kontrast kolorów
C. odpowiedni kontrast tekstu
D. dostępność dla technologii wspomagających

Które zasady w UI wynikają z Gestalt ?

A. zwiększanie liczby elementów dla lepszej percepcji
B. brak zależności między układem a interpretacją treści
C. grupowanie elementów poprzez bliskość
D. podobieństwo wizualne jako wskazówka relacji