

# Program szkolenia:

## Modern Agile

### Informacje:

<b>Nazwa:</b>	<b>Modern Agile</b>
<b>Kod:</b>	<b>Agile-people</b>
<b>Kategoria:</b>	Agile Software Development
<b>Odbiorcy:</b>	hr, admini, developerzy, testerzy, management, architekci, analitycy, DevOps, Product Owners, Scrum Masters
<b>Czas trwania:</b>	2 dni
<b>Forma:</b>	80% wykłady / 20% warsztaty

---

Prawie 20 lat po narodzinach Agile Software Development coś poszło nie tak.

Zatraciliśmy zasady i wartości stojące za konkretnymi implementacjami manifestu.

Praktyki menedżerskie wzięły górę nad technicznymi.

Przywiązujemy za dużą wagę do certyfikacji i zbieramy kolejne pieczętki jak odznaki harcerskie.

Kopiujemy praktyki Scrumowe z lat 90, model Spotify z 2012, modne rozwiązania z Doliny Krzemowej zapominając, że pracujemy w systemach złożonych, gdzie najlepsze praktyki nie działają.

To szkolenie pokazuje jak wypracować własne kontekstowe praktyki w oparciu o zasady Modern Agile:

- Najpierw zapewnij bezpieczeństwo
- Make people awesome
- Eksperymentuj i ucz się szybko
- Stale dostarczaj wartość

W czasie szkolenia nauczysz się:

- Traktować przyjmowanie nowych kolegów i koleżanek do pracy jako pełnowartościową pracę pierwszej kategorii, a nie przykry obowiązek.
- Zapewniać swoim współpracownikom poczucie bezpieczeństwa w codziennej pracy i częściej zauważać gdy go brakuje.
- Wspierać budowanie umiejętności eksperckich wśród swoich współpracowników i rozpoznawać co może blokować szybsze postępy.
- Rozpoznawać dlaczego niektórym osobom trudno razem pracować i jak dostosować swój przekaz do różnych poziomów zaawansowania.
- Minimalizować wpływ błędów poznawczych przy trudnych wyborach i podejmować odrobinę lepsze decyzje technologiczne czy rekrutacyjne.
- Podnosić odpowiedzialność i zaangażowanie zespołu w realizację celów bez komenderowania i kontroli.
- Projektować środowisko pracy pod kątem responsywności a nie optymalizacji kosztowej.

## BO·TT·EGA

IT minds

- Kwestionować mainstreamowe praktyki, role, rytuały, gdy te stały się częścią problemu, zamiast być częścią rozwiązania.

### Zalety szkolenia:

- Principle-driven a nie Framework-driven
- Bazuje na badaniach zgodnych z metodą naukową
- Skupiamy się na kontekście projektu informatycznego i pracy developerów, bez analogii typowych dla miękkich szkoleń
- Trener jest biegły w praktykach menadżerskich i technicznych oraz nie posiada certyfikatów wydanych przez oligarchów Scrumowych

## Szczegółowy program:

### 1. Intro

- 1.1. Agile Software Development nie tak jak miało być
- 1.2. Cel Software Developmentu
- 1.3. "Best practices" a Model Cynefin
- 1.4. Praktyki = Zasady(Kontekst)
- 1.5. Koszt utraconych możliwości
- 1.6. Jasne Punkty

### 2. Najpierw zapewnij bezpieczeństwo

- 2.1. nawyk kluczowy bezpieczeństwa
- 2.2. Kontekst: kod legacy
  - 2.2.1. strategiczne podejście z behawioralną analizą kodu
- 2.3. Kontekst: wdrożenia
  - 2.3.1. antywzorzec: bohater wdrożeniowy
  - 2.3.2. antywzorzec: armia wdrożeniowa
  - 2.3.3. teatr zarządzania ryzykiem
- 2.4. Kontekst: reakcja na awarie
  - 2.4.1. błędy poznawcze i kultura obwiniania
  - 2.4.2. post-mortem bez obwiniania
- 2.5. Kontekst: komunikacja w zespole
  - 2.5.1. psychological safety
  - 2.5.2. siła słabej komunikacji
  - 2.5.3. rola adwokata diabła
  - 2.5.4. aktywne słuchanie i charyzma uwagi

**3. Make People Awesome**

## 3.1. Twórcy

## 3.1.1. Zasada postępu

## 3.1.2. Job crafting

## 3.2. Użytkownicy

## 3.2.1. zapomnij o stakeholders

## 3.3. Uczniowie

## 3.3.1. od myślenia wolnego do szybkiego

## 3.3.2. teoria i praktyka pożądanych trudności

## 3.4. Przyszli eksperci

## 3.4.1. celowe i zamierzone ćwiczenia

## 3.4.2. głęboka praca w IT

## 3.4.3. szybkość i jakość informacji zwrotnej od ludzi i systemów

## 3.4.4. modelowanie ekspertów

## 3.4.5. Model braci Dreyfus w kontekście parowania

## 3.5. Ciężko pracujący

## 3.5.1. dobry sen &gt; praktyki techniczne

## 3.5.2. "crunch mode"

## 3.5.3. problemy z macierzami kompetencji

## 3.6. Nowy zespół

## 3.6.1. Ćwiczenie Druckera

## 3.7. Nowi pracownicy

## 3.7.1. Serdeczne powitanie

## 3.7.2. Lokalizowanie ekspertów

## 3.8. Odchodzący pracownicy

3.8.1. Serdeczne pożegnanie

3.9. Pozostali pracownicy

3.9.1. Symulacja utraty wiedzy

3.10. Potencjalni kandydaci

3.10.1. antywzorce w ofertach pracy

3.10.2. przykładowa oferta

## 4. Eksperymentuj i ucz się szybko

4.1. Kontekst: Wybór technologii

4.1.1. multitracking

4.1.2. ADR

4.1.3. Efekt Lindy

4.2. Kontekst: Rekrutacja

4.2.1. skracanie procesu

4.2.2. próbka realistycznej pracy

4.2.3. rodzaje dowodów

4.2.4. feedback dla i od kandydatki

4.3. Kontekst: Planowanie roadmapy

4.3.1. #noprojects

4.3.2. mityczne "business value"

4.3.3. mapowanie impaktu

4.3.4. dewelopment sterowany hipotezami

4.4. Kontekst: Rozpoczynanie nowych projektów

4.4.1. Inception Deck

## 5. Stale dostarczaj wartość

5.1. Kontekst: dostarczanie oprogramowania

5.1.1. Continuous Delivery edycja matematyczna

5.1.2. Przewrotne zachęty

5.1.3. 4 metryki Continuous Delivery

5.2. Kontekst: integracja kodu

5.2.1. Test Certyfikacyjny Continuous Integration

5.2.2. Trunk Based Development a Build Feature Branching

5.3. Kontekst: struktura organizacji

5.3.1. Mapowanie strumienia wartości

5.3.2. Prawo Conway'a

5.3.3. Zespoły zorientowane na aktywność

5.3.4. Zespoły zorientowane na wynik/produkt/value stream

5.3.5. Zespoły wpierające: enabling team, complicated subsystem team, platform team

5.3.6. Rozmiar zespołu

5.4. Kontekst: komunikacja pomiędzy zespołami

5.4.1. współpraca, "as-a-service" i facylitacja

5.4.2. sieci społecznościowe w kodzie

5.5. Kontekst: własność kodu

5.5.1. rozmycie odpowiedzialności

5.5.2. pomiary potrzeby koordynacji w kodzie

5.6. Kontekst: obciążenie pracą

5.6.1. slack i mit zajętości

5.6.2. teoria ograniczeń

5.7. Kontekst: ewolucja zespołu

5.7.1. mit stabilnych zespołów

