BO·TT·EGA
IT minds

Program szkolenia:

Trust Framework for AI-Assisted Development

Informacje:

Nazwa:	**Trust Framework for AI-Assisted Development**
Kod:	**ai-trust**
Kategoria:	AI
Odbiorcy:	architekci, developerzy
Czas trwania:	3 dni
Forma:	20% wykłady / 80% warsztaty hands-on

Przestań być wąskim gardłem — zbuduj bramki jakości, które pozwolą Ci zaufać kodowi z AI bez czytania każdej linijki.

LLM-y generują kod szybciej niż jakikolwiek człowiek jest w stanie go zweryfikować. To jednocześnie problem i szansa. Ludzkie code review się nie skaluje. Jeśli w Twoim zespole zaufanie kodowi (szczególnie LLMowemu) wymaga, żeby ktoś czytał każdą linijkę, nigdy nie będziesz szybszy niż najwolniejszy reviewer w zespole. Płacisz za prędkość AI, ale dostajesz przepustowość ludzkiego review.

Rozwiązaniem są bramki jakości (quality gates) — warstwy weryfikacji operujące powyżej kodu. Czy spełnia specyfikację? Czy scenariusze behawioralne przechodzą? Czy CI jest zielone na trunku? Czy telemetria produkcyjna potwierdza oczekiwane zachowanie po deployu? Każda bramka buduje zaufanie bez konieczności czytania implementacji. Razem pozwalają Twojemu zespołowi wykorzystać pełną prędkość generowania kodu przez LLM-y.

LLM-y nie różnią się aż tak bardzo od ludzkich developerów. Popełniają błędy, robią skróty, gubią edge case'y i dryfują od intencji — tak samo jak ludzie, tylko szybciej. Bramki jakości potrzebne do zaufania outputowi LLM-ów to dokładnie te same, które inżynierowie budowali przez ostatnie dekady, żeby ufać sobie nawzajem. Specyfikacje, testy, CI, feature flagi, monitoring — nic z tego nie jest nowe. AI po prostu sprawia, że staje się to niezbędne.

Grupa docelowa:

Seniorzy, tech leadzi i architekci w zespołach, które już korzystają z narzędzi AI do kodowania (Claude Code, Copilot, Cursor) i wiedzą, że ludzkie review nie skaluje się do tempa kodu generowanego przez AI. Potrzebują bramek jakości, które pozwolą zespołowi wykorzystać pełną prędkość LLM-ów zamiast wąskiego gardła na ręcznym review.

Wymagania wstępne:

- 3+ lata doświadczenia w profesjonalnym programowaniu
- Aktywne korzystanie z przynajmniej jednego narzędzia AI do kodowania
- Znajomość TypeScript/JavaScript (inne języki możliwe — do uzgodnienia przed szkoleniem)
- Podstawowa znajomość testowania (testy jednostkowe, CI)
- Dostęp do narzędzia AI do kodowania (Claude Code, Cursor, Copilot, itp. — do uzgodnienia przed szkoleniem)

BO·TT·EGA
IT minds

Co wyniesiesz ze szkolenia:

- Zidentyfikujesz i nazwiesz konkretne zagrożenia związane z zaufaniem do kodu i artefaktów generowanych przez LLM-y
- Połączysz sprawdzone praktyki inżynieryjne z każdym zagrożeniem jako konkretne środki zaradcze
- Napiszesz specyfikacje będące weryfikowalnymi kontraktami dla kodu generowanego przez AI
- Użyjesz diagramów C4 do komunikacji intencji architektonicznych narzędziom AI
- Zastosujesz test-first weryfikację behawioralną — komunikujesz intencję, weryfikujesz przez testy behawioralne (nie przez code review)
- Przećwiczysz trunk-based development z krótkimi branchami dla pracy z AI
- Wdrożysz feature flagi jako siatkę bezpieczeństwa dla kodu AI na produkcji
- Skonfigurujesz monitoring z automatycznym wyłączaniem flag przy anomaliach
- Połączysz wszystkie praktyki w powtarzalny framework zaufania dla swojego zespołu
- Zaimplementujesz realną funkcjonalność używając pełnego frameworka — bez czytania ani jednej linii kodu produkcyjnego

Zalety szkolenia:

- Zbudujesz bramki jakości, które zastąpią ręczne review automatyczną weryfikacją
- Poznasz które praktyki inżynieryjne (wiele z nich ma dekady) bezpośrednio odpowiadają na zagrożenia LLM-ów
- Użyjesz Spec-driven development rozszerzony o diagramy architektury C4 do komunikacji wizji do AI
- Nauczysz się komunikować intencję przez specyfikacje, i zweryfikować implementację przez testy behawioralne — nie przez code review
- Wykorzystasz Trunk-based development, feature flagi i monitoring jako ujednoliconą siatkę bezpieczeństwa
- Skonfigurujesz automatyczne wyłączanie flag przy wykryciu anomalii aby chronić produkcję w sposób zautomatyzowany
- Finalny eksperyment: zaimplementujesz realną funkcjonalność bez czytania ani jednej linii kodu produkcyjnego wygenerowanego przez AI

BOTTEGA IT minds

Szczegółowy program:

1. Krajobraz zagrożeń — co może pójść nie tak z kodem z LLM-a

1.1. Taksonomia błędów kodu z LLM-ów: halucynowane API, dryf logiki, ciche podatności bezpieczeństwa

1.2. Kategorie zagrożeń: poprawność, bezpieczeństwo, architektura, testowanie, dryf

1.3. Dlaczego ludzkie review się nie skaluje — i co je zastępuje

1.4. Budowanie wspólnego modelu zagrożeń dla Twojego zespołu

2. Praktyki inżynieryjne jako środki zaradcze

2.1. Eksploracja: które sprawdzone praktyki odpowiadają na które zagrożenia?

2.2. Warsztat: Mapowanie wieloletnich metodyk inżynieryjnych na zdetekowane zagrożenia LLM-ów

3. Właściwe narzędzie do AI-assisted development

3.1. Krajobraz narzędzi AI do kodowania: chat, autocomplete, IDE, agent CLI

3.2. Dlaczego agenci CLI najlepiej integrują się z bramkami jakości (testy, CI, trunk-based flow)

3.3. Ocena Twojego obecnego narzędzia pod kątem wymagań frameworka zaufania

3.4. Zarządzanie kontekstem, reużywalne skille i persystentna konfiguracja projektu w różnych narzędziach

4. Claude Code — szybki start

4.1. Pierwsze kroki z Claude Code (lub OpenCode, Aider)

4.2. CLAUDE.md: trwała pamięć Twojego projektu

4.3. Zarządzanie kontekstem i zapobieganie context poisoning

4.4. Warsztat: Stwórz swój pierwszy CLAUDE.md

5. Spec-Driven Development z C4

5.1. Specyfikacje jako weryfikowalne kontrakty dla kodu generowanego przez AI

5.2. Diagramy architektury C4 do komunikacji wizji z AI

5.3. Jak specyfikacja + kontekst C4 staje się kontraktem, który AI musi spełnić

BOTTEGA IT minds

5.4. Warsztat: Napisz specyfikacje i diagramy C4 ograniczające wykonanie AI

6. Weryfikacja behawioralna — test-first, behavior-driven

6.1. Dyscyplina test-first: testy behawioralne definiują kontrakt przed rozpoczęciem implementacji

6.2. Weryfikacja implementacji przez wyniki testów behawioralnych — nie przez code review

6.3. Testy weryfikujące zachowanie, nie implementację — czytelne i ujawniające intencję

6.4. Praktyczne kontrakty behawioralne — nie ceremonialny Gherkin

6.5. Scenariusze data-driven weryfikujące output AI bez czytania implementacji

6.6. Warsztat: Od specyfikacji przez testy behawioralne do wygenerowanej i zweryfikowanej implementacji

7. Trunk-Based Development dla zespołów pracujących z AI

7.1. Krótkie branche i ciągła integracja ze zmianami generowanymi przez AI

7.2. Małe merge'e utrzymujące wkład AI w stanie przejrzystym i odwracalnym

7.3. Strategie branchowania dopasowane do tempa rozwoju z AI

7.4. Warsztat: Trunk-based workflow z AI-assisted feature development

8. Feature Flags — deploy bez strachu

8.1. Feature flagi jako siatka bezpieczeństwa dla kodu AI na produkcji

8.2. Stopniowy rollout, natychmiastowy rollback

8.3. Oddzielenie deploymentu od release'u

8.4. Warsztat: Implementacja feature flag ze strategiami rollout i rollback

9. Monitoring kodu AI na produkcji

9.1. Observability i alerting zamykające pętlę zaufania po deployu

9.2. Definiowanie warunków anomalii powiązanych z feature flagami

9.3. Automatyczne wyłączanie flag — produkcja chroni się sama, gdy kod AI się źle zachowuje

9.4. Warsztat: Konfiguracja monitoringu z automatycznym wyłączaniem flag przy anomaliach

BO·TT·EGA
IT minds

10. Dowód — zbuduj swój framework, potem dostarcz bez czytania kodu

10.1. Łączenie wszystkich praktyk w powtarzalny framework zaufania dla Twojego zespołu

10.2. Dostosowanie frameworka do codebase'u i workflow Twojego zespołu

10.3. Implementacja realnej zmiany z wykorzystaniem każdej warstwy frameworka

10.4. Nadzoruj specyfikacje, testy, pipeline i monitoring — nie kod produkcyjny

10.5. Bez czytania ani jednej linii implementacji wygenerowanej przez AI

10.6. Warsztat: Zaprojektuj swój framework zaufania, potem udowodnij że działa — end-to-end delivery funkcjonalności