BO·TT·EGA
IT minds

92%[1] **Adopcja AI**
w przedsiębiorstwach

55%[1] **Szybkość realizacji**
pojedynczego zadania

+40%[2] **Wzrost podatności bezpieczeństwa**
w kodzie AI

+39%[3] **Krótkoterminowy churn kodu**
kod AI przerabiany w ciągu 14 dni

-50%[3] **Wskaźnik refaktoryzacji**
dług technologiczny od AI

-7.5%[4] **Spadek jakości systemu**
przez wąskie gardło PR review

Źródła:
[1] GitHub & Microsoft Research (500+ inż. Fortune 500 · ankieta A/B 95 programistów)
[2] Stanford University — programiści z asystentami AI piszą kod zawierający o ok. 40% więcej krytycznych luk (SQL Injection, niepoprawne szyfrowanie) i mają fałszywe przekonanie o jego poprawności.
[3] GitClear Analysis — telemetria 153 mln LOC: kod AI masowo usuwany/poprawiany w ciągu 14 dni (churn) oraz unikanie refaktoryzacji → narastanie długu technologicznego (złamanie zasady DRY).
[4] Google Cloud DORA Report — spadek jakości i wzrost Change Failure Rate spowodowany zalewaniem procesu Code Review kodem AI niskiej jakości (rubber-stamping); 76–80% zespołów integruje AI w co najmniej jednym etapie SDLC.
[5] Gartner — 85% projektów AI/ML w przedsiębiorstwach nie przynosi oczekiwanego zwrotu z inwestycji.

EXECUTIVE STRATEGY BRIEFING

BUDOWANIE FABRYKI OPROGRAMOWANIA

Ramy architektoniczne, podział pracy i „okiełznanie" AI przy wytwarzaniu systemów klasy enterprise

Postępująca industrializacja wytwarzania oprogramowania

Przenosi środek ciężkości z kodowania na:
Internal Developer Platform (IDP),
wydajną kolaborację **Business–Engineering–AI**,
oraz nową dyscyplinę **AI Harness Engineering**.

Odbiorcy: Dyrektorzy / Zarząd
Data: 01 września 2026
Przygotowane przez: Bottega IT Minds — Transformacje AI

PARADOKS PRODUKTYWNOŚCI AI

Gdzie naprawdę leży wąskie gardło?

Obecna rzeczywistość rynkowa potwierdzona przez szereg badań:

- wysoka adopcja AI (92%),
- wzrost szybkości pracy deweloperów (+55%),
- brak przełożenia na realny wynik / sukces biznesowy organizacji.

Według Teorii Ograniczeń (TOC) lokalne usprawnienie pojedynczego ogniwa nie zwiększa przepustowości całego systemu, a jedynie wywołuje presję na kolejne etapy łańcucha wartości, ujawniając nowe wąskie gardła.

Ograniczenie 0 PRZED AI

Tempo pisania kodu

Głównym ograniczeniem była wydajność inżynierów. Asystenci AI wyeliminowali je na poziomie jednostki (programisty).

↓

Ograniczenie 1 OBECNY STAN

Jakość i bezpieczeństwo

Zalew generowanego kodu zderza się z niewydajnym procesem zarządzania jakością (bazującym na ludzkim doświadczeniu i skupieniu na detalu). W konsekwencji otrzymujemy paraliż kontroli jakości lub lawinowy dług technologiczny (Stanford: +40% luk bezpieczeństwa, GitClear: −50% refaktoryzacji).

↓

Ograniczenie 2 KOLEJNY ETAP

Przepływ wiedzy i wdrożenia

Barierą staje się transfer wiedzy dziedzinowej Biznes–Inżynieria oraz brak dojrzałości procesów wdrożeniowych: wartość utyka przez niedokładne lub źle zrozumiane wymagania.

↓

Ograniczenie 3 DOCELOWY PUNKT CIĘŻKOŚCI

Świadomość rynku i kreowanie wartości

Przy zautomatyzowanej, bezpiecznej fabryce ostatecznym ograniczeniem jest trafna identyfikacja potrzeb rynkowych i przekuwanie ich w produkty z realnym ROI oraz iterowanie w odpowiednim kierunku.

Dane: 92% i +55% — GitHub & Microsoft Research [1]; +40% luk — Stanford [2]; −50% refaktoryzacji — GitClear [3].

STRUKTURA BRIEFINGU

Systemowa odpowiedź na ograniczenia

Wytwarzanie kodu stało się tanie i powszechne. Odblokowanie Ograniczenia 0 pokazało, że optymalizacja IT wyłącznie przez dostarczanie AI opiera się na błędnym założeniu, że problemem był czas pisania kodu.

O ile pisanie kodu jest szybkie, o tyle:

- dobrze zidentyfikowane potrzeby (biznesu, klientów, rynku, regulatorów itp.),
- dobrze postawione hipotezy, jak je właściwie zaadresować,
- dobrze określone wymagania niefunkcjonalne (technologiczne, architektoniczne itp.),
- dobrej jakości kod, który spełnia wszystkie powyższe punkty,
- dowody jakości i zgodności kodu

już nie są tak proste, szybkie i tanie do osiągnięcia.

STRUKTURA RAPORTU

AI Harness Engineering
Automatyzacja bramek jakości i bezpieczeństwa → natychmiastowa odpowiedź na Ograniczenie 1.

IDP i AI SDLC
Standaryzacja procesów i architektury → systemowa odpowiedź na Ograniczenie 1.

Reorientacja ról (FDE)
Udrożnienie przepływu wiedzy i skupienie inżynierów na wartości biznesowej → odpowiedź na Ograniczenie 2.

Definition of Value (DoV)
Skupienie na twardych wskaźnikach i potwierdzaniu strategii → odpowiedź na Ograniczenie 3.

NOWY PARADYGMAT

Industrializacja inżynierii oprogramowania

Dotychczasowy model wytwarzania oprogramowania, oparty na manualnym „szyciu" kodu przez zespoły programistów, ustępuje miejsca zautomatyzowanej fabryce oprogramowania. AI umożliwia masową produkcję kodu, lecz stwarza nowe zagrożenia i wymaga nowych ról, procesów oraz nowych narzędzi.

Kluczową rolę w wytwarzaniu oprogramowania wysokiej jakości od zawsze pełnili **wykwalifikowani ludzie** oraz **platforma (IDP)**. W dobie AI: **Harness Engineering** jest nową kluczową dyscypliną, ponieważ nawet kompetentni i zmotywowani ludzie nie są już w stanie „ręcznie" zagwarantować wszystkich cech oczekiwanych od oprogramowania.

Cel strategiczny Software Factory

Stworzenie środowiska, w którym funkcjonalności biznesowe są dostarczane szybciej i taniej, w sposób powtarzalny, domknięty i bezpieczny — **przy zachowaniu obecnej jakości lub jej poprawieniu**, bez ryzyka degradacji istniejących systemów.

KRYTYCZNE WYZWANIE

Uprząż / kaganiec na AI

AI bez ram to dług technologiczny (Ograniczenie 1)

Statystyki są jednoznaczne: **+40% krytycznych luk bezpieczeństwa** (Stanford), **+39% porzucanego kodu** (GitClear) oraz **−50% refaktoryzacji**. W ujęciu TOC to bezpośredni efekt odblokowania **Ograniczenia 0** (tempa pisania kodu) bez zabezpieczenia **Ograniczenia 1** (kontroli jakości). Deweloperzy, zachęceni pozorną szybkością, dopisują nowe bloki kodu zamiast przemyśleć i porządkować istniejący kod — dług technologiczny rośnie w rekordowym tempie.

Harness

Harness (z ang. uprząż), w kontekście pracy z AI, to zestaw narzędzi, które gwarantują (lub, jeśli to niemożliwe, maksymalizują prawdopodobieństwo) wystąpienia określonej cechy w procesie wytwórczym czy w samym wytwarzanym oprogramowaniu, np. specyfikacja jest spisana i spełnia Definition of Ready przed implementacją, implementacja zapewnia kompatybilność wsteczną z API, brak znanych podatności bezpieczeństwa w zależnościach, poprawność architektoniczną, pokrycie testami 100% biznesowej logiki aplikacji, testy e2e dla scenariuszy akceptacyjnych itp.

Klucz do sukcesu wykorzystania AI

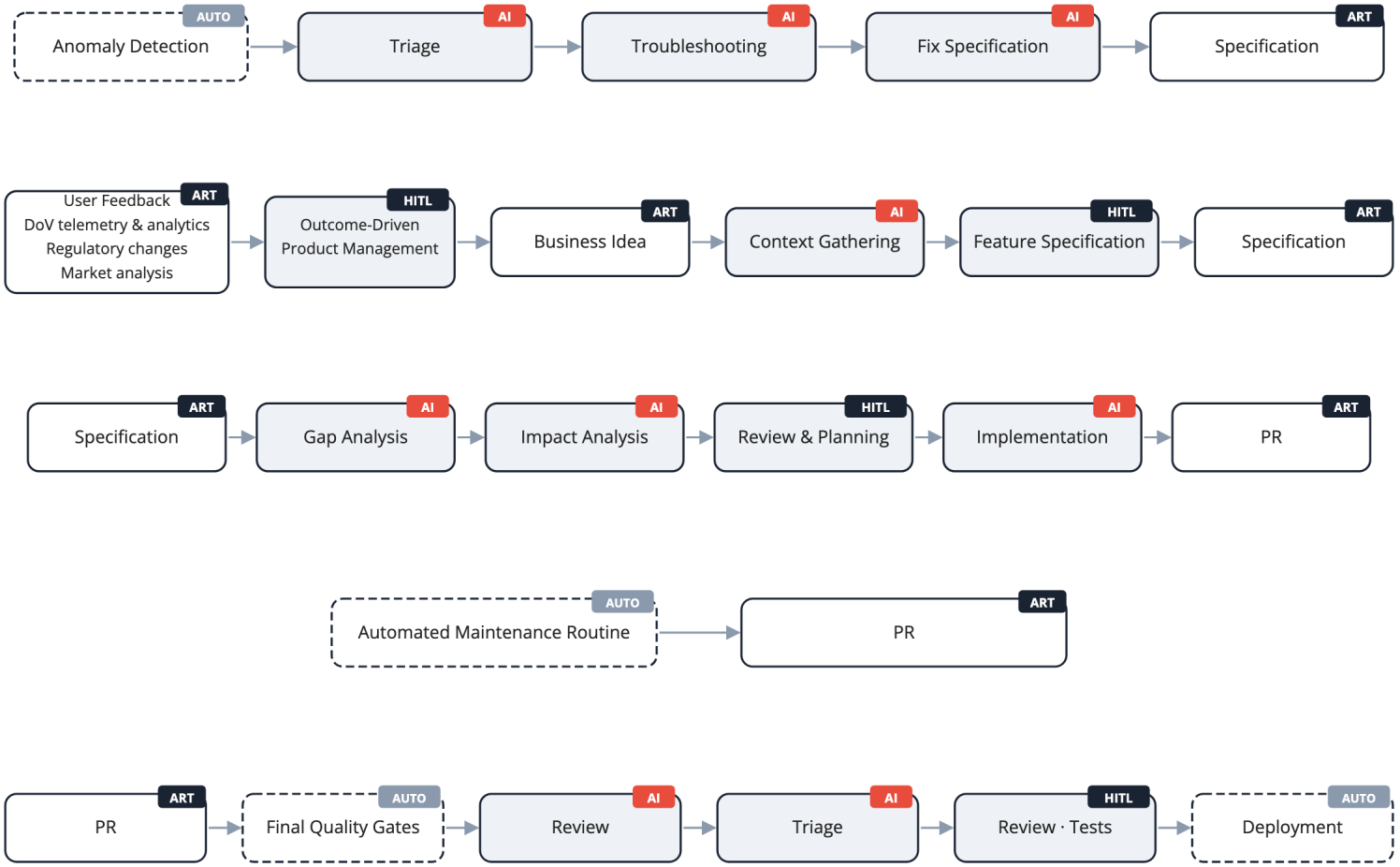
Harness Engineering to narzucenie na modele generatywne bezwzględnego rygoru architektonicznego, rygorów walidacyjnych, obserwowalności (Observability) i zgodności z wymogami (compliance). AI musi produkować kod w sposób możliwie deterministyczny, stabilny i zgodny ze standardami organizacji.
Zautomatyzowanie dowodów: **Definition of Ready, Definition of Done, Definition of Released, Definition of Live, Definition of Value**.

Źródła: [2] Stanford University — +40% krytycznych luk bezpieczeństwa; [3] GitClear — +39% porzucanego kodu i −50% refaktoryzacji.

AI SDLC

Wsparcie AI dla każdego etapu cyklu życia oprogramowania

Przykładowe przepływy pracy:



HITL (Human In The Loop) interaktywna praca człowieka i AI
AUTO — proces automatyczny
AI — autonomiczna praca AI, określana jako AFK (Away From Keyboard)
ART — artefakt (np. ticket, specyfikacja, PR)
każdy krok wymaga własnych bramek jakościowych i może spowodować potencjalną eskalację do człowieka

SOFTWARE FACTORY

Kluczowe komponenty wspierające AI SDLC

AI Harness NEW
instructions, SKILLS, CLI, MCP, quality gates itd.

AI Sandbox NEW
bezpieczna przestrzeń dla AI — zapewniająca bezpieczeństwo i zgodność agentów

Cloud/OnPrem Development Environment NEW
hostowane powtarzalne środowiska pracy dla AI i dostępne dla ludzi

Internal Developer Platform (IDP)
wewnętrzna platforma deweloperska — self-service dla zespołów i agentów

Internal Knowledgebases, API Management
bazy wiedzy dziedzinowej i zarządzanie API

SDLC Tooling & Delivery
GitHub, GitLab, CI, Jira, Nexus itd.

Environments & Monitoring
stabilne: test, UAT, prod oraz adhoc/ephemeral

Security & Compliance Tools
narzędzia i dashboardy bezpieczeństwa i zgodności

NEW — komponenty nowe lub rzadko występujące jak do tej pory.
Pozostałe to elementy istniejące, które powinny już istnieć w dojrzałej organizacji.

KAPITAŁ LUDZKI

Trzy archetypy i ewolucja ról inżynieryjnych

Google Cloud DORA raportuje **7,5-procentowy spadek jakości systemów** i wzrost awaryjności wdrożeń — przez **paraliż Code Review** i powierzchowne zatwierdzanie kodu AI (rubber-stamping). Gwałtowny rozwój narzędzi generatywnych uwypuklił podział kadry na trzy postawy.

Programista biorący odpowiedzialność

Elita wytwórcza — biznesowy inżynier end-to-end

Nawet jeśli dostaje zwięzłe zadanie („jednolinijkowiec w Jirze"), to samodzielnie prowadzi cały proces: zbiera wiedzę w organizacji, rozmawia z ekspertami dziedzinowymi, weryfikuje hipotezy biznesowe, tworzy rozwiązanie, testuje i dostarcza „na gotowe".

W tej grupie występują zarówno fani, jak i sceptycy AI.

Programista odtwórczy (koder)

Oczekuje precyzyjnej specyfikacji, brak refleksji

Wymaga mikrozarządzania i łopatologicznych specyfikacji. Nie zna i nie szuka kontekstu biznesowego. Skupiony wyłącznie na implementacji (i szybkim „commit, push"), co generuje ukryte awarie.

W tej grupie często występuje entuzjazm do AI — przyspiesza ono tworzenie kodu niskiej jakości z precyzyjnych specyfikacji.

Technik / Hacker / Architekt

Wysoce wydajny, ale bez kontekstu biznesowego

Pasjonat technologii budujący oprogramowanie wysokiej jakości (technicznie), ale niskiej wartości biznesowej. Nieznający żargonu biznesowego, nieporadny w rozmowach z biznesem.

W tej grupie często obserwujemy entuzjazm połączony z ostrożnością.

Ewolucja roli programisty:

- **Programista biorący odpowiedzialność:** najbardziej pożądane i wartościowe jednostki w dobie AI. Ich wartość rośnie, gdy opanują Platformę i AI i skupią się na dostarczaniu wartości biznesowej (**Forward Deployed Engineering**) — dziś bywają uwikłani w mikrozarządzanie odtwórcami i walkę o jakość.
- **Programista odtwórczy (koder):** ta grupa musi dostać jasny sygnał zmian, kierunek rozwoju oraz czas i wsparcie, by zaczęła brać odpowiedzialność. Czasem wystarczy dać przestrzeń i pozwolić popełniać błędy — dojrzałość może pojawić się odruchowo.
- **Technik / Hacker / Architekt:** to jednostki dające wartość w każdym zespole oraz dobrzy kandydaci do **zespołu platformowego**. Trzeba jednak dbać o odpowiedni mix kompetencyjny i charakterologiczny w każdym zespole — zespół złożony wyłącznie z takich osób na pewno dowiezie coś „wspaniałego", ale niekoniecznie to, czego potrzebuje organizacja.

Warto zwrócić uwagę na role (→ odpowiedź na Ograniczenie 2):

- **Forward Deployed Engineers (Inżynierowie produktu):** Odpowiadają za dowiezienie wartości biznesowej, co często wymaga interakcji z interesariuszami, analitykami, biznesem / użytkownikami, prawnikami. W tej roli niezbędne jest zrozumienie domeny biznesowej i odbiorców, weryfikacja hipotez biznesowych, specyfikacja pod AI oraz **dostarczenie rozwiązania przy pomocy AI** i **wzięcie odpowiedzialności** za wynik.
- **Platform Team (Architekci maszyn):** Odpowiedzialni za projektowanie, utrzymanie i optymalizację „maszyny wytwórczej", architektur referencyjnych, CI/CD, infrastruktury, instrukcji, skilli, CLI, MCP itd. dla AI oraz, co obecnie najważniejsze, **bramek jakościowych (Quality Gates)**.

Relacja Platform Team ↔ Forward Deployed Engineer (FDE)

Częstą patologią jest izolacja zespołu platformowego budującego narzędzia „dla samego siebie". W prawidłowym modelu **Platform Team jest podwykonawcą i dostawcą wewnętrznych narzędzi dla FDE** — FDE wyznaczają potrzeby, a Platform Team je realizuje w modelu **Platform-as-a-Product**.

Źródło: [4] Google Cloud DORA — 7,5-procentowy spadek jakości systemów i wzrost Change Failure Rate (rubber-stamping).

ZALEGŁOŚCI

Nadrobienie zaległości w Platformie i SDLC

Transformacja AI to okazja do naprawienia zaległości i wyznaczenia najlepszych standardów. Dzięki prawidłowemu wykorzystaniu AI każda inwestycja w **platformę**, standaryzację procesów i architektur oraz w **ludzi** zwraca się natychmiastowo.

Wydajny strumień wiedzy dziedzinowej

Programiści odizolowani od biznesu, długi proces pętli zwrotnych.

Internal Developer Platform

Platforma oderwana od realiów lub projekty nieaktualizowane do obecnych standardów platformy.

Bazy wiedzy & Kontekst dla AI

Nieudokumentowana architektura, istotne regulacje, dokumentacja dla użytkownika czy ogólna wiedza biznesowa organizacji.

Automatyzacja jakości

Niewystarczające bramki jakościowe i bezpieczeństwa (security) wbudowane w zautomatyzowany potok CI/CD.

Architektury i technologie legacy

Kod, w którym rozwój utknął przez błędne zarządzanie złożonością na poziomie architektury i kodu, lub systemy w technologiach jak COBOL.

REKOMENDACJE

Rekomendowane akcje

- **Bezpieczeństwo i compliance agentów AI**
  Narzucenie bezwzględnego wymogu uruchamiania agentów AI w odizolowanych środowiskach klasy MicroVM z rygorystyczną filtracją ruchu wyjściowego (egress). Uruchamianie agentów AI na maszynach deweloperskich lub w kontenerach Docker naraża organizację na ataki lub przypadkowy wyciek własności intelektualnej, tokenów i kluczy dostępowych.
- **Inwestycja w Wewnętrzną Platformę Deweloperską (IDP)**
  Urealnienie platformy / synchronizacja z realiami produktów, konsekwentne rozwijanie IDP w modelu Platform-as-a-Product. Inwestycja skróci czas wprowadzenia produktów na rynek (Time-to-Market), obniży koszty infrastruktury i stworzy standaryzowane środowisko dla ludzi oraz agentów AI.
- **Transformacja modelu talentów i reorientacja ról**
  Uruchomienie programu transformacji kompetencji oraz dostosowanie polityki rekrutacji i wynagrodzeń w celu przejścia od odtwórczego kodowania do inżynierii zorientowanej na wartość biznesową. Decyzja ta podnosi efektywność kosztową kadr, zapobiega utracie samodzielności zespołów i zwiększa retencję kluczowych talentów.
- **Przejście z mierzenia ilości (Output) na wartość biznesową (Outcome)**
  Reorganizacja wskaźników KPI/OKR pionu technologii poprzez wdrożenie rygorystycznych kryteriów Definition of Value. Gwarantuje to, że masowa produkcja kodu przy pomocy AI przełoży się bezpośrednio na cele strategiczne i finansowe firmy, eliminując ryzyko tworzenia niepotrzebnego oprogramowania.

BO·TT·EGA
IT minds

> *Efektywności IT nie mierzy się wolumenem wygenerowanego kodu ani zużyciem tokenów AI (Outputs). Z perspektywy biznesowej tokeny i linijki kodu stanowią koszt oraz przyszłe zobowiązanie (dług technologiczny), które należy minimalizować.*

Miarą sukcesu IT powinna być głównie realizacja celów biznesowych i strategicznych (Outcomes), które organizacja musi w sposób ciągły wyznaczać, komunikować i monitorować.

Wskaźniki komercyjne, przykładowo:

- wzrost przychodów z kanałów cyfrowych,
- wyższa wartość klienta (LTV),
- redukcja wskaźnika odpływu klientów (Churn Rate).

Cele strategiczne, przykładowo:

- stabilne utrzymanie i zapewnienie zgodności,
- dostosowanie produktu do nowego rynku czy segmentu,
- zgodność z nadchodzącymi wymogami regulacyjnymi (Regulatory Compliance & Risk Mitigation).

Należy jednak pamiętać, że powyższe miary to wielowymiarowe wskaźniki wynikowe (lagging indicators). Ich realizacja nie zależy wyłącznie od działu IT, lecz jest wypadkową wysiłku całej organizacji, aktualnych uwarunkowań rynkowych oraz dynamiki konkurencyjnej.

MIARA EFEKTYWNOŚCI IT

Metryki jakości procesu wytwórczego

Dla uzyskania pełnego obrazu konieczne jest równoległe stosowanie wskaźników bezpośrednich (leading indicators), znajdujących się w sferze wyłącznej kontroli i odpowiedzialności pionu IT. Stanowią je metryki jakości i dojrzałości procesu wytwórczego oprogramowania:

Metryki defektów i efektywności wykrywania, przykładowo:

- skuteczność wyłapywania błędów przed produkcją (Defect Removal Efficiency),
- odsetek usterek przenikających na produkcję (Defect Leakage),
- wskaźnik zadań zaliczających weryfikację za pierwszym razem (First Time Pass Rate),
- częstotliwość ponownego otwierania rozwiązanych zgłoszeń (Defect Reopen Rate).

Metryki dostarczania i niezawodności (DORA), przykładowo:

- częstotliwość bezpiecznych wdrożeń produkcyjnych (Deployment Frequency),
- czas przejścia kodu od commitu do uruchomienia na produkcji (Lead Time for Changes),
- odsetek wdrożeń generujących awarię lub wymagających hotfixa (Change Failure Rate),
- średni czas przywrócenia pełnej sprawności po awarii (MTTR).

Metryki przepływu i wydajności operacyjnej, przykładowo:

- stosunek czasu aktywnej pracy nad zadaniem do przestojów i oczekiwania zadania (Flow Efficiency),
- czas recenzowania kodu oraz wskaźnik odrzuconych przeglądów (PR Turnaround & Bounce Rate),
- udział prac powtórzonych wynikających ze złych specyfikacji (Re-work Rate),
- poziom przeciążenia zespołów pracą w toku (WIP Overload).

Fundamenty Harness Engineeringu

- wyznacz cele,
- wskaż jak je osiągnąć,
- ściśle zdefiniuj, jak je zmierzyć (jak udowodnić Outcome).

Tym razem aplikowane nie na AI w konsoli programisty,
lecz nakładane przez zarząd na **Wewnętrzną Fabrykę Oprogramowania** (Business-Engineering-AI)
→ odpowiedź na Ograniczenie 3.

Nasi **Klienci**

Jako inżynierowie, za najważniejszy wskaźnik naszego sukcesu uważamy rosnącą liczbę powracających klientów.

ABB · BOSCH · orange · Credit Suisse
Ericsson · Fujitsu · ING · intel
Lufthansa · Motorola · Nokia · Roche
SAP · T-Mobile · TomTom · Volvo

OBSZARY KOMPETENCJI:

Fintech, e-commerce, telco, motoryzacja, bankowość mobilna, systemy ERP, systemy CMS, systemy zarządzania danymi, start-upy, wojsko, energetyka, półprzewodniki, przemysł 4.0

Interesuje Cię, co Bottega może dla Ciebie zrobić? **Skontaktuj się z nami.**

Iwona Sobótka — Koordynatorka Konsulting: +48 500 286 698
Biuro: +48 81 473 29 44, contact@bottega.com.pl