# Ustrój agentowy SDLC: od autonomii maszyn do odpowiedzialności człowieka w świetle Code Revealed Alexa Cassaniego


**AUTOR**

Fundacja Dobre Państwo



**STRESZCZENIE / ABSTRACT**

Artykuł analizuje ewolucję sztucznej inteligencji w inżynierii oprogramowania, wskazując na przejście od generatywnych asystentów do autonomicznych agentów AI. Autor podkreśla, że kluczowa zmiana nie dotyczy ilości produkowanego kodu, lecz sprawczości i zdolności osiągania celów w złożonym środowisku. Na przykładzie benchmarku SWE-bench oraz koncepcji Agent-Computer Interface (ACI) wykazano, że efektywność AI zależy nie tylko od modelu, ale od narzędzi zaprojektowanych specyficznie dla maszyn. Tekst redefiniuje podział pracy poznawczej w procesie SDLC, przesuwając punkt ciężkości z prostego promptowania na zarządzanie trajektorią działań agenta oraz nadzór człowieka nad finalnym wdrożeniem.

8. The article analyzes the evolution of artificial intelligence in software engineering, pointing to the shift from generative assistants to autonomous AI agents. The author emphasizes that the key change is not about the volume of code produced, but about agency and the ability to achieve goals within a complex environment. Using the SWE-bench benchmark and the Agent-Computer Interface (ACI) concept, it is demonstrated that AI efficiency depends not only on the model but on tools designed specifically for machines. The text redefines the division of cognitive labor in the SDLC process, shifting the focus from simple prompting to managing the agent's trajectory of actions and human oversight over final deployment.


Artykuł analizuje fundamentalną zmianę paradygmatu w inżynierii oprogramowania: przejście od prostego generowania kodu przez AI do systemów agentowych, które

wykazują realną sprawczość w środowisku technicznym. Autor stawia tezę, że drastyczny spadek kosztu wytworzenia linii kodu nie oznacza wzrostu produktywności, lecz przesuwa punkt ciężkości wartości z rzemiosła pisania na zarządzanie kontekstem, nadzór i odpowiedzialność za systemy. Tekst argumentuje, że bez kompleksowej architektury nadzoru (governance) i kalibracji autonomii maszyn, AI staje się jedynie „wzmacniaczem chaosu", generującym tzw. dług epistemiczny – sytuację, w której organizacja posiada działający system, którego nikt już nie rozumie. Kluczowym wnioskiem jest konieczność przedefiniowania roli programisty z wykonawcy na orkiestratora oraz przejścia od rachunku kosztów tokenów do analizy całkowitego kosztu akceptacji zmiany (total cost of accepted change), gdzie najwyższą wartością staje się nie ilość kodu, lecz zdolność do jego selekcji i redukcji.

**SŁOWA KLUCZOWE**

ustrój agentowy SDLC | Agent AI | Generative AI | SWE-bench | Agent-Computer Interface (ACI) | inżynieria kontekstu | Vibe Coding | Agent-Supervised Development | AI-Assisted Coding | gradient delegowanej sprawczości | dług techniczny AI | net cognitive savings | return on deletion | agent-centric SDLC | human verification tax

## Od generowania kodu do sprawczej inżynierii

Od maszyny piszącej kod do maszyny uczestniczącej w działaniu. Największym błędem poznawczym w opisie sztucznej inteligencji w inżynierii oprogramowania jest rozpoczynanie analizy od pytania, ile kodu potrafi dziś wygenerować maszyna. Pytanie to było zasadne w epoce pierwszych asystentów programistycznych, lecz szybko stało się anachroniczne. Mierzy bowiem rewolucję miarą poprzedniego świata.

Podobnie jak sensu fabryki nie da się pojąć poprzez zliczenie ruchów ręką oszczędzonych pojedynczemu rzemieślnikowi, tak znaczenia agentów AI nie sposób sprowadzić do liczby znaków, funkcji czy klas napisanych bez udziału klawiatury. Zmienia się bowiem nie tylko sposób wytwarzania kodu, ale sam podział pracy poznawczej: kto definiuje zadanie, interpretuje kontekst i proponuje rozwiązanie; kto wykonuje czynności, kontroluje odchylenia oraz testuje rezultat i – co

kluczowe – kto ostatecznie ponosi odpowiedzialność za wdrożenie zmiany na produkcję. Materiał źródłowy Cassaniego trafnie lokuje ten przełom między Generative AI a Agent AI.

Generatywna sztuczna inteligencja reaguje przede wszystkim na bodziec: użytkownik zadaje pytanie, model generuje odpowiedź, a człowiek przenosi rezultat do środowiska działania. Agent wprowadza natomiast dodatkową warstwę sprawczości. Może otrzymać cel, rozbić go na sekwencję czynności, korzystać z narzędzi, przeszukiwać repozytorium, edytować pliki i uruchamiać testy, by na ich podstawie podejmować kolejne kroki. W miejsce relacji „pytanie-odpowiedź" pojawia się trajektoria. Jest to różnica głębsza, niż sugerowałoby samo słowo „agent", ponieważ informacja zostaje tu sprzężona z możliwością realnej ingerencji w środowisko. To właśnie w tym punkcie teoria spotkała się z doświadczeniem badawczym.

SWE-bench powstał jako odpowiedź na oczywistą niewystarczalność benchmarków opartych na produkowaniu izolowanych fragmentów kodu. Zamiast prosić model o napisanie funkcji w sterylnym laboratorium, badacze zestawili 2294 rzeczywiste problemy z poziomu zgłoszeń i odpowiadających im pull requestów z dwunastu popularnych repozytoriów Pythona. Zadaniem modelu stało się zrozumienie problemu oraz wprowadzenie odpowiedniej zmiany w istniejącej bazie kodu. Pierwotny wynik Claude 2, wynoszący 1,96%, brutalnie obnażył przepaść między umiejętnością generowania poprawnie wyglądających instrukcji a zdolnością do wykonywania rzeczywistej pracy inżynieryjnej.

Odkrycie to ma znaczenie wykraczające poza informatykę. Kompetencja nie jest sumą odpowiedzi na pytania. Kompetencja jest zdolnością osiągania celu w środowisku stawiającym opór. Program produkcyjny to historia decyzji, zależności i kompromisów; to zbiór nieaktualnych konwencji, wyjątków, zabezpieczeń i śladów pracy wielu ludzi. W notatce trafnie przywołano skalę SWE-bench: kontekst problemu może obejmować setki tysięcy linii kodu, podczas gdy poprawna naprawa wymaga zmiany zaledwie niewielkiego fragmentu, często w kilku plarach jednocześnie. Dlatego programowanie agentowe jest problemem nie tyle generacji, co lokalizacji, selekcji, sekwencjonowania i kontroli.

SWE-agent wykonał kolejny istotny ruch intelektualny. Badacze odwrócili tradycyjne pytanie: zamiast ulepszać wyłącznie model, zapytali, czy narzędzia, którymi kazano mu się posługiwać, nie zostały przypadkiem zaprojektowane dla zupełnie innego gatunku użytkownika. Człowiek w IDE korzysta ze wzroku, pamięci przestrzennej, kolorów i intuicji wypracowanej przez tysiące godzin pracy z interfejsem. Model językowy nie posiada takiej aparatury poznawczej. Stąd pojęcie Agent-Computer Interface (ACI) – interfejsu projektowanego świadomie dla agenta, a nie dla człowieka.

Rezultat jest konceptualnie donioślejszy niż kolejna poprawa wyniku w benchmarku. Choć SWE-agent osiągnął 12,5% pass@1 na SWE-bench, najważniejszy wniosek brzmi inaczej: inteligencja systemu jest funkcją nie tylko modelu, ale i środowiska, w którym ten model operuje. Notatka źródłowa rozwija tę intuicję poprzez zasadę zwięzłego feedbacku, specjalizowane narzędzia do przeglądu plików, ograniczanie szumu informacyjnego, lintery oraz rollback. W dokumentacji SWE-agent podkreśla się m.in. wykorzystanie dedykowanego file viewera i mechanizmu lintowania, który blokuje syntaktycznie błędne edycje.

## Od promptowania do architektury agentowej: zarządzanie kontekstem i środowiskiem

Można odnieść wrażenie, że historia informatyki zatoczyła osobliwe koło. Przez dziesięciolecia projektowaliśmy human-computer interaction, ucząc maszyny, jak być wygodnymi dla człowieka. Agentowa sztuczna inteligencja zmusza nas teraz do stworzenia drugiej ergonomii – machine-computer interaction. W świecie ludzkim dobry interfejs ukrywa złożoność maszyny przed użytkownikiem; w świecie agentowym dobry interfejs odcina od modelu tę część złożoności środowiska, która nie jest niezbędna do podjęcia kolejnej poprawnej decyzji. To zasada niemal epistemologiczna: więcej informacji nie zawsze oznacza więcej wiedzy. Nieograniczony kontekst może stać się nie biblioteką, lecz śmietnikiem.

Stąd jeden z najciekawszych wątków analizy: paradoks kontekstu. Wraz ze wzrostem ilości nieustrukturyzowanego materiału model może tracić zdolność skupienia się na przyczynowym centrum problemu. Podważa to popularne przekonanie, że wystarczy "dać AI dostęp do wszystkiego". Dostęp nie jest jeszcze rozumieniem. Bibliotekarz nie staje się mądrzejszy tylko dlatego, że otwarto przed nim wszystkie magazyny naraz.

W projektowaniu agentów pojawia się zatem nowa funkcja intelektualna – inżynieria kontekstu: świadome decydowanie, jaka wiedza, w jakiej chwili, w jakiej reprezentacji i z jakim stopniem autorytetu powinna znaleźć się w polu decyzyjnym modelu. W tym punkcie wyraźnie zaznacza się różnica pomiędzy klasycznym prompt engineeringiem a architekturą agentową. Prompt jest komunikatem; agent jest systemem działania. Prompt może być znakomity, a proces katastrofalny. Można bowiem świetnie poinstruować model, który otrzymał nadmierne uprawnienia, błędny kontekst lub dostęp do krytycznej infrastruktury bez mechanizmu cofania skutków. Z tego powodu obiektem projektowania przestaje być pojedyncza "rozmowa z AI". Projektuje się pętlę działania: intencję, plan, dostęp do narzędzi, granice operacyjne, obserwację rezultatów, testy, możliwość eskalacji i punkt, w którym maszyna musi oddać decyzję człowiekowi.

SWE-Gym dodaje do tego trzeci element. Jeśli SWE-bench odpowiada na pytanie "czy agent potrafi wykonać rzeczywistą pracę?", a SWE-agent pyta "jakiego środowiska działania potrzebuje?", to SWE-Gym skupia się na kwestii: "jak można go takiej pracy systematycznie nauczyć?". Środowisko obejmuje 2438 rzeczywistych zadań programistycznych z aktywnymi środowiskami i testami; wykorzystano je do trenowania agentów oraz weryfikatorów, osiągając w eksperymentach wyniki rzędu 32% na SWE-bench Verified i 26% na SWE-bench Lite dla otwartych modeli. Notatka słusznie interpretuje tę triadę jako przejście od interfejsu, przez trening, do ewaluacji. Nie należy jednak traktować tych liczb jak tablic Mojżeszowych agentowej rewolucji. Benchmark jest instrumentem pomiarowym, a nie samą rzeczywistością.

Późniejsza praca SWE-Bench+ zakwestionowała część wyników, wskazując na ryzyko ujawniania rozwiązań w treściach issues oraz niewystarczającą siłę niektórych testów; po odfiltrowaniu tych przypadków wyniki dla SWE-Agent+GPT-4 okazały się znacznie niższe. Nie unieważnia to jednak SWE-bench – przeciwnie, pokazuje proces dojrzewania nauki. Dobra nauka nie chroni benchmarku przed krytyką. Dobra nauka benchmark poddaje benchmarkowaniu.

Z tych badań płynie pierwsza zasada transformacji SDLC: nie istnieje "wydajność AI" w oderwaniu od architektury działania. Model, który w jednym środowisku ponosi seryjne porażki, po zmianie interfejsu może osiągać znacznie lepsze rezultaty. Odpowiedni trening na trajektoriach interakcji pozwala ujawnić kompetencje, których model nie wykazywał jako zwykły generator kodu. Z kolei nawet najsilniejszy model zostanie unieszkodliwiony przez przeładowany kontekst, wadliwe narzędzia lub błędne testy. W tej perspektywie pytanie przedsiębiorstwa "który model kupić?" przypomina pytanie XIX-wiecznej fabryki o to, jaki silnik parowy wybrać, zadawane bez namysłu nad układem hali, transmisją napędu, kompetencjami robotników czy kontrolą jakości.

## AI jako wzmacniacz chaosu i przesunięcie wartości ku nadzorowi

Potwierdzają to badania DORA. Raport z 2024 roku wskazuje na jednoczesną korelację adopcji AI z poprawą jakości dokumentacji, kodu i szybkości code review, przy jednoczesnym negatywnym wpływie na stabilność oraz throughput dostarczania. Szacuje się, że wzrost adopcji o 25% wiązał się między innymi ze spadkiem stabilności dostarczania o 7,2%.

Nie jest to dowód na to, że AI „pogarsza programowanie”, lecz znacznie ciekawszy sygnał: lokalna optymalizacja może destabilizować system jako całość. Jeśli jeden etap rurociągu zaczyna generować więcej kodu, podczas gdy review, testowanie, integracja, bezpieczeństwo i deployment

pozostają zorganizowane według dawnej przepustowości, powstaje zator. Maszyna zwiększa prędkość wejścia, ale organizacja nie nadąża z przyswajaniem zmian. Rok później DORA sformułowała tę obserwację w sposób jeszcze dojrzalszy.

Opierając się na odpowiedziach blisko pięciu tysięcy profesjonalistów oraz materiale jakościowym, badacze opisali AI jako amplifier – wzmacniacz cech istniejącej organizacji. Silny system organizacyjny dzięki AI może stawać się silniejszy; chaos wyposażony w szybszą generację staje się natomiast szybszym chaosem. W tym jednym spostrzeżeniu kryje się niemal cała polityczna ekonomia adopcji agentów. Paradoks produktywności opisany w notatce nie jest więc drobnym zastrzeżeniem wobec entuzjazmu, lecz centralnym problemem transformacji.

W starym modelu produktywność programisty intuicyjnie wiązano z tempem tworzenia użytecznego kodu. W modelu agentowym koszt napisania kolejnej linijki może gwałtownie spaść, przez co przestaje być ona dobrem rzadkim. Dobrem rzadkim staje się zdolność rozpoznania, która linia w ogóle powinna istnieć. Niedoborem stają się uwaga seniora, poprawna architektura, kontekst domenowy, wiarygodny test, bezpieczeństwo, rozliczalność oraz umiejętność odróżnienia rozwiązania pięknego od takiego, które tylko pięknie wygląda.

Dlatego materiał Cassaniego proponuje przesunięcie roli programisty w stronę orkiestratora i superwizora. Człowiek ma przewodzić, weryfikować i integrować; agent zaś wykonywać coraz większą część pracy operacyjnej w ramach nadanych mu granic. Nie jest to jednak prosty „awans" do bardziej prestiżowej pracy, lecz także utrata części dawnego monopolu zawodowego. Pisanie kodu, przez dekady będące materialnym dowodem kompetencji, może zostać zdegradowane z istoty zawodu do jednej z technik realizacji celu. To, co dawniej stanowiło rzemiosło, staje się surowcem. A gdy surowiec tanieje, wartość przenosi się ku projektowaniu, selekcji, odpowiedzialności i decyzji. Właśnie dlatego kultura „vibe coding" jest jednocześnie fascynująca i niebezpieczna.

Język naturalny obniża koszt przejścia od pomysłu do prototypu, zbliżając autora potrzeby do jej technicznej materializacji. Notatka słusznie ogranicza ten paradygmat do eksploracji, MVP i Proof-of-Concept, ostrzegając przed bezpośrednim przenoszeniem tak powstałego kodu do produkcji. Demokracja tworzenia oprogramowania nie znosi bowiem praw inżynierii. Most zaprojektowany łatwiej nie staje się przez to odporniejszy na grawitację. Aplikacja powstała w sobotni wieczór na podstawie pięciu promptów może wyglądać cudownie na ekranie, a jednocześnie nosić w sobie zalążek poniedziałkowej katastrofy.

Przyszłość nie będzie zatem prostą opozycją „programista albo AI". Taki spór przypomina dziewiętnastowieczny lęk przed zastąpieniem robotnika przez maszynę, pomijając fakt, że maszyny przede wszystkim reorganizowały system produkcji, kompetencje, własność i zarządzanie. Bardziej

prawdopodobnym punktem ciężkości jest system hybrydowy: maszyny produkują coraz więcej możliwych działań, a ludzie budują instytucje pozwalające odróżnić działanie dopuszczalne od niedopuszczalnego, produktywne od pozornie produktywnego i autonomię użyteczną od nadmiernej. W tym miejscu kończy się niewinna historia o „pomocniku do programowania", a zaczyna historia o ustroju agentowym SDLC.

Jeżeli agent może planować, wykonywać, sprawdzać i poprawiać własne działania, przedsiębiorstwo musi zdecydować nie tylko, co maszyna potrafi dokonać, lecz czego wolno jej dokonać, gdzie ma się zatrzymać, komu złożyć raport, jakie dowody pozostawić oraz kto ma prawo pociągnąć cyfrowy sznur bezpieczeństwa. Następnym krokiem nie będzie więc katalog narzędzi.

## SDLC jako gradient delegowanej sprawczości

Rozwiązaniem będzie rekonstrukcja całego cyklu SDLC jako gradientu delegowanej sprawczości – od Vibe Coding, przez Agent-Supervised Development, aż po ścisłe AI-Assisted Coding w obszarach, w których człowiek pozostaje nie ceremonialnym podpisem pod decyzją maszyny, lecz jej rzeczywistym suwerenem. Mówiąc wprost: SDLC staje się gradientem delegowanej sprawczości, rozpiętym między „vibe" a pełną odpowiedzialnością.

Pierwsza faza rewolucji AI w programowaniu polegała na przyspieszeniu pisania kodu. Druga jest znacznie bardziej wymagająca intelektualnie: dotyczy przeprojektowania granicy między ludzką decyzją a maszynowym wykonaniem. Klasyczny Software Development Life Cycle wyrósł z epoki, w której poszczególne fazy – wymagania, projektowanie, implementacja, testowanie, wdrożenie i utrzymanie – można było analitycznie od siebie oddzielić. Każda z nich miała swoich specjalistów, dedykowaną dokumentację, punkty przekazania odpowiedzialności i rytuały kontroli. Agentowa sztuczna inteligencja te granice rozmywa.

Model może podczas analizy wymagań zaproponować architekturę, w fazie implementacji wygenerować test, podczas testowania odkryć konieczność refaktoryzacji, a w trakcie utrzymania samodzielnie przygotować poprawkę. Cassani określa tę konwergencję mianem agent-centric SDLC: fazy nie tyle znikają, co zaczynają na siebie nachodzić i funkcjonować równolegle. Zmiana ta, choć pozornie techniczna, w istocie jest zmianą ustrojową. Dotychczasowy SDLC organizował kolejność czynności; agent-centric SDLC musi organizować także hierarchię uprawnień.

Trzeba bowiem odpowiedzieć na pytanie, które tradycyjna inżynieria oprogramowania stawiała rzadko: jak wiele działania można oddelegować wykonawcy, który nie jest osobą i nie posiada odpowiedzialności prawnej, operuje z prędkością nieosiągalną dla człowieka, a jego zdolność do

przekonującego formułowania odpowiedzi nie idzie w parze ze zdolnością do trafnego rozpoznawania konsekwencji? Właśnie dlatego pojęciem organizującym dojrzałą adopcję AI nie powinna być maksymalizacja autonomii, lecz jej kalibracja.

Cassani proponuje trzy podstawowe paradygmaty interakcji: Vibe Coding, Agent-Supervised Development oraz AI-Assisted Coding. W materiale źródłowym przypisano im orientacyjnie wysoką, średnią i niską autonomię AI, wskazując odmienne obszary zastosowania: od prototypów i MVP, przez standardowy rozwój i refaktoryzację, aż po produkcję i kod krytyczny. Samych wartości procentowych autonomii nie należy jednak fetyszyzować – nie istnieje przecież termometr, który pozwoliłby obiektywnie stwierdzić, że agent posiada akurat 58 procent samodzielności. Wartość tej typologii tkwi gdzie indziej: swoboda maszyny powinna być funkcją konsekwencji jej błędu.

Prowadzi to do zasady, którą można nazwać prawem odwrotnej proporcjonalności autonomii i kosztu nieodwracalności. Im tańszy jest błąd, łatwiejszy rollback i mniejszy zakres oddziaływania zmiany, tym więcej swobody można przyznać agentowi. Im bardziej decyzja dotyczy danych produkcyjnych, finansów, bezpieczeństwa, uprawnień użytkowników, integralności infrastruktury lub zobowiązań prawnych, tym mniej miejsca powinno pozostawać na autonomiczne eksperymentowanie.

Nie wynika to z założenia, że człowiek jest zawsze mądrzejszy od modelu – taka teza byłaby równie naiwna, co wiara w nieomylność AI. Powodem jest asymetria odpowiedzialności: maszyna może wywołać skutek, ale organizacja musi umieć ten skutek wyjaśnić, odtworzyć, naprawić i przypisać do decyzji, którą ktoś miał prawo podjąć.

## Vibe Coding jako laboratorium szybkich prototypów

Na najbardziej liberalnym krańcu spektrum znajduje się Vibe Coding. To przestrzeń szybkiego R&D, Proof-of-Concept i eksperymentowania, w której liczy się maksymalna szybkość iteracji, kod może mieć charakter jednorazowy, a Execution Plan oraz Logbook są radykalnie uproszczone. Jednocześnie należy sformułować kategoryczne zastrzeżenie wobec bezpośredniego przenoszenia takiego kodu na produkcję ze względu na dług techniczny i niską stabilność strukturalną. Jest w tym pewna historyczna ironia. Przez dziesięciolecia informatyka próbowała nauczyć człowieka języka maszyny. Vibe Coding odwraca wektor: maszyna zaczyna rozumieć wystarczająco dużo języka człowieka, by materializować jego zamiar bez tradycyjnego aktu programowania. To właśnie tłumaczy kulturową atrakcyjność tego paradygmatu.

Twórca nie musi już przeprowadzać idei przez wąskie gardło składni, bibliotek i frameworków. Może powiedzieć: chcę zobaczyć taki formularz, taki przepływ, taki model aplikacji – i obserwować pojawiający się rezultat. Programista w tej fazie przypomina mniej murarza, a bardziej scenografa, reżysera albo architekta szkicującego warianty przestrzeni.

Koszt eksperymentu spada, a wraz z nim rośnie liczba prób. Schumpeterowska "twórcza destrukcja" otrzymuje tutaj narzędzie nie tyle niszczenia istniejących przedsiębiorstw, ile obniżenia ceny samego aktu próbowania. Niski koszt próby jest jednak błogosławieństwem tylko do chwili, w której nie zostanie pomylony z niskim kosztem błędu. Prototyp może powstać w godzinę, lecz system zbudowany z prototypowych kompromisów może przez lata pobierać od organizacji podatek w postaci długu technicznego. Łatwość tworzenia jest niebezpieczna właśnie dlatego, że odbiera błędowi jego dawny sygnał ostrzegawczy – wysoki koszt wykonania. Kiedy napisanie tysiąca nowych linii kodu wymagało wielu dni pracy, sama cena produkcji zmuszała do zastanowienia się, czy są one w ogóle potrzebne. Kiedy podobną ilość kodu można wygenerować w kilka minut, hamulec ekonomiczny znika. Pojawia się zatem nowe zadanie governance: skoro kod stał się tani, koszt jego dopuszczania do systemu musi zostać ponownie urealniony przez kontrolę jakości. Vibe Coding powinien być więc rozumiany jako laboratorium, a nie konstytucja systemu produkcyjnego.

Jego naturalnym prawem jest możliwość wyrzucenia rezultatu do kosza. "Disposable code" nie jest obelgą; w odpowiednim miejscu stanowi świadomie zakupioną opcję poznawczą. Kod pomaga odpowiedzieć na pytanie, czy pomysł działa, czy użytkownik go rozumie i czy architektura jest wykonalna. Problem zaczyna się dopiero wtedy, gdy organizacja zakochuje się we własnym eksperymencie i uznaje, że skoro działa na ekranie, to może już zostać. Prowizorka informatyczna, podobnie jak prowizorka instytucjonalna, posiada niezwykły talent do długowieczności.

## Agent-Supervised Development jako model zarządzania delegowaną sprawczością

Środkowy obszar spektrum – Agent-Supervised Development – jest z tej perspektywy znacznie ciekawszy, gdyż to właśnie tutaj może wykształcić się codzienny model pracy hybrydowego zespołu. W tym podejściu agent otrzymuje znaczną samodzielność wykonawczą w zadaniach średniej złożoności, przy rozwijaniu modułów, większych refaktoryzacjach czy zmianach obejmujących wiele plików, jednak porusza się on między uprzednio zdefiniowanymi punktami kontrolnymi. Execution Plan wyznacza drogę, Supervision Checkpoints dzielą ją na etapy, a Logbook dokumentuje przebieg prac. Konstrukcja ta opiera się na zasadzie znanej ze starszych systemów zarządzania ryzykiem: deleguj wykonanie, ale nie deleguj bezwarunkowo prawa do przekraczania granic. Dobry kapitan

nie trzyma dłoni marynarza przy każdym ruchu steru, lecz definiuje kurs, granice bezpieczeństwa i sytuacje wymagające wezwania na mostek. Chirurg nie wykonuje osobiście wszystkich czynności zespołu operacyjnego, ale istnieją momenty, w których decyzja nie może zostać zdecentralizowana. Kontroler ruchu lotniczego nie prowadzi samolotu, lecz organizuje przestrzeń, w której autonomia pilotów pozostaje bezpieczna.

Agent-Supervised Development przenosi tę logikę do świata maszyn poznawczych. Execution Plan staje się w tym systemie czymś więcej niż techniczną notatką – jest kontraktem epistemicznym pomiędzy człowiekiem a agentem. Zanim powstanie kosztowny rezultat, strony (używając metafory, gdyż agent nie jest stroną w sensie prawnym) uzgadniają, jak maszyna interpretuje cel, jakie elementy systemu zamierza zmienić, jaką kolejność działań przewiduje i gdzie dostrzega ryzyko. Plan musi poprzedzać rozpoczęcie zasadniczej pracy, a Logbook ma później rejestrować decyzje, problemy i odstępstwa. Takie podejście eliminuje jeden z najbardziej kosztownych nawyków tradycyjnego zarządzania projektem: poprawianie nieporozumienia dopiero po jego materializacji. Błędna interpretacja zadania wykryta w pięciu zdaniach planu jest tania; ta sama interpretacja rozlana po trzydziestu plikach, migracji danych i testach staje się niezwykle kosztowna. Stara maksyma budowniczych – „dwa razy mierz, raz tnij” – zyskuje nowe życie w świecie, w którym samo cięcie stało się niemal darmowe. Paradoksalnie, im szybsza staje się maszyna, tym cenniejsza jest pauza przed działaniem.

Logbook pełni funkcję przeciwną i komplementarną. Plan mówi: „zamierzam”, dziennik zaś: „zrobiłem”. Różnica między tymi dokumentami pozwala odkryć tzw. dryf. Agent mógł natrafić na nieprzewidzianą zależność, zmienić metodę, utworzyć dodatkowy plik lub zastosować bibliotekę, której wcześniej nie przewidywał. W klasycznej pracy programisty podobne decyzje bywają ukryte w pamięci autora. W pracy agentowej jest to szczególnie ryzykowne, ponieważ organizacja może otrzymać rezultat bez człowieka zdolnego odtworzyć drogę jego powstania. Logbook staje się więc nie dekoracją compliance, lecz narzędziem przeciwdziałania ciemnej wiedzy – kodowi, który działa, choć nikt już naprawdę nie wie dlaczego. W tym kontekście pojawia się jedna z najistotniejszych maksym koncepcji Cassaniego: nie powinno się scalać kodu, którego zespół nie potrafi wyjaśnić i utrzymać. Zasada ta powinna zostać podniesiona do rangi kryterium dojrzałości organizacyjnej.

## Dług epistemiczny wymusza portfel reżimów autonomii

W epoce przedagentowej niezrozumiały kod powstawał zazwyczaj wskutek złej dokumentacji, rotacji pracowników lub technicznej brawury pojedynczego twórcy. W erze agentów niezrozumiałość może być produkowana przemysłowo: maszyna tworzy rozwiązania szybciej, niż

zespół jest w stanie je przyswoić. Wówczas przedsiębiorstwo staje się właścicielem systemu, nad którym stopniowo traci władzę poznawczą. Pojawia się tu nowa odmiana zadłużenia, którą można nazwać długiem epistemicznym. Podczas gdy dług techniczny oznacza, że system działa dzisiaj kosztem trudniejszej zmiany jutro, dług epistemiczny oznacza, że system działa dzisiaj kosztem coraz słabszego rozumienia, dlaczego w ogóle działa. Pierwszy obciąża architekturę; drugi uderza w zdolność organizacji do podejmowania decyzji. W świecie agentów oba te długi mogą rosnąć równocześnie, a łatwość generowania kolejnych rozwiązań może przez długi czas maskować problem. Maszyna naprawia coś, czego człowiek już nie rozumie, następnie inna maszyna koryguje konsekwencje poprzedniej interwencji i organizacja zaczyna przypominać państwo, w którym każdą nieczytelną ustawę próbuje się naprawić nową, jeszcze mniej czytelną regulacją.

Supervision Checkpoints są próbą przerwania tego procesu. Ich idea jest prosta: autonomii nie przyznaje się agentowi na cały projekt jednorazowo, lecz w interwałach – od checkpointu do checkpointu. Źródło proponuje przykładowo osobne punkty kontrolne dla architektury, baz danych, dostawców i testów. Po każdym etapie człowiek odzyskuje możliwość oceny nie tylko tego, czy wykonano zadanie, ale także czy dalszy kierunek pozostaje racjonalny. Jest to kluczowa różnica między kontrolą a audytem. Audyt potrafi wskazać po katastrofie, kto i kiedy popełnił błąd; checkpoint ma sprawić, by część katastrof w ogóle nie nastąpiła.

Na drugim krańcu znajduje się AI-Assisted Coding – model najmniej spektakularny, a przez to niezwykle ważny. W materiale Cassaniego jest on przeznaczony dla kodu produkcyjnego o wysokich wymaganiach stabilności; AI działa tam jako Pair Programmer, podczas gdy człowiek zachowuje ścisłą kontrolę nad zmianą. Plan oraz Logbook stają się formalnymi elementami procesu, a wygenerowany rezultat podlega przeglądowi przed scaleniem. Paradygmat ten lokowany jest szczególnie blisko systemów bezpieczeństwa, płatności oraz utrzymania produkcji. Można uznać to za krok wstecz: skoro agent potrafi coraz więcej, dlaczego ograniczać go do roli pomocnika?

Odpowiedź ujawnia dojrzałość całej architektury. Autonomia nie jest nagrodą przyznawaną inteligentniejszej maszynie, lecz uprawnieniem nadanym procesowi, który potrafi bezpiecznie ponosić skutki jej działania. Sam wzrost zdolności modelu nie uzasadnia więc automatycznego zwiększenia zakresu uprawnień. Potrzebny jest równoległy wzrost obserwowalności, jakości testów, niezawodności rollbacku, kontroli dostępu, rozliczalności i zdolności zespołu do walidowania wyniku. Z tej perspektywy organizacja powinna posiadać nie jedną politykę AI, lecz portfel reżimów autonomii. Cassani proponuje tutaj RACM (Rapid AI Capability Map) – dynamiczne mapowanie narzędzi według zadań SDLC, możliwego poziomu autonomii oraz niezawodności i jakości rezultatów. Koncepcja ta przeciwstawia się typowemu błędowi adopcji technologicznej: zakupiliśmy

narzędzie, więc teraz szukamy jak największej liczby miejsc, w których można je użyć. RACM odwraca tę logikę.

Najpierw definiujemy zadanie, jego ryzyko i wymagany poziom odpowiedzialności; dopiero potem wybieramy model, interfejs i zakres uprawnień. Nie pytamy więc: „Czy nasz agent potrafi modyfikować bazę danych?", lecz: „W jakich warunkach organizacja jest gotowa pozwolić mu modyfikować bazę danych?". Nie pytamy: „Czy model potrafi dokonać code review?", lecz: „Jakiego rodzaju błędy wykrywa niezawodnie, jakich systematycznie nie widzi i kto odpowiada za te niewykryte?".

## Rozróżnienie między autonomią techniczną a decyzyjną w delegowaniu zadań AI

Nie pytamy już tylko: „Czy AI potrafi projektować architekturę?". Pytamy raczej: „Które elementy decyzji architektonicznej są odwracalne, które tworzą wieloletni lock-in i na jakim poziomie decyzja musi pozostać ludzkim zobowiązaniem?". To przejście od pytania o capability do pytania o delegability stanowi jeden z kluczowych kroków w stronę dojrzałości organizacji. Logikę tę rozwija PAIP (Practical AI Integration Patterns), wyróżniając pięć wzorców: eksplorację i prototypowanie, supervised assistance, agent-supervised development, automatyzację zadań powtarzalnych oraz AI-assisted security controls.

Istota tych wzorców nie tkwi w samej klasyfikacji. PAIP pozwala dostrzec, że autonomia techniczna i autonomia decyzyjna to dwie różne kwestie. Agent może otrzymać szerokie prawo do wykonywania operacji mechanicznych, o ile reguły są jednoznaczne, a skutki łatwe do opanowania. Ten sam agent powinien mieć jednak minimalne uprawnienia w przypadku decyzji rzadkich w wykonaniu, lecz strategicznych w konsekwencjach. Dobrym przykładem tej różnicy jest automatyzacja zadań powtarzalnych. Formatowanie, generowanie fragmentów dokumentacji czy określone czynności CI/CD mogą być silnie zautomatyzowane właśnie dlatego, że przestrzeń decyzji jest tu niewielka. W takim układzie kontrola zostaje zakodowana w samym procesie, a Logbook może przybrać formę prostego rejestru wykonania. Maszyna robi bardzo dużo, ale decyduje o bardzo mało.

Taki model jest często bezpieczniejszy niż pozornie skromny asystent, który podejmuje jedną, lecz strategicznie doniosłą decyzję w oparciu o niepełny kontekst. Podobną ostrożność źródło zachowuje wobec AI-Assisted Security Controls. Agent może znakomicie skalować wyszukiwanie znanych podatności i analizę zależności, ale znacznie gorzej radzi sobie z unikalną logiką biznesową, gdzie

bezpieczeństwo zależy od znaczenia, a nie tylko od syntaktycznego wzorca. Komputer od dawna przewyższa człowieka w przeszukiwaniu miliardów możliwości, lecz nie oznacza to, że lepiej rozumie społeczny sens każdej z nich.

Im bardziej problem staje się domenowy, kontekstualny, polityczny lub aksjologiczny, tym mniej wystarcza czysta zdolność rozpoznawania wzorców. Na tym tle nabiera znaczenia zasada Cassaniego: „automatyzuj, aby wzmacniać, a nie zastępować". Materiał źródłowy wiąże ją z delegowaniem maszynom zadań powtarzalnych przy jednoczesnym pozostawieniu człowiekowi kontroli nad architekturą, strategią produktu i relacjami z interesariuszami.

## Nadzór to realna sprawczość, a nie rytualna akceptacja

Tę maksymę nie należy rozumieć sentymentalnie, jako deklaracji, że żadne stanowisko nie zostanie zastąpione przez AI. Historia automatyzacji nie daje podstaw do takiego komfortu. Sens tej zasady jest głębszy: system powinien automatyzować przede wszystkim tam, gdzie maszyna zwiększa zdolność człowieka do podejmowania lepszych decyzji, zamiast budować organizację, w której człowiek podpisuje rezultaty, których już nie rozumie. W ten sposób zmienia się także znaczenie Human-in-the-Loop. HITL nie może oznaczać obecności człowieka jako rytualnej pieczątki.

Człowiek, który otrzymuje pięćset zmian i ma trzy minuty na kliknięcie „approve", formalnie pozostaje w pętli, lecz faktycznie został z niej usunięty. Nadzór istnieje tylko wtedy, gdy osoba dysponuje czasem, kompetencjami, informacjami i realną możliwością odmowy. W przeciwnym razie human-in-the-loop staje się human-on-the-form: człowiek nie sprawuje władzy nad procesem, lecz jedynie użycza mu nazwiska. Dlatego przyszły SDLC nie powinien być budowany jako wyścig ku „pełnej autonomii". Taka metafora zakłada, że maksimum samodzielności jest naturalnym punktem dojścia, niczym najwyższy poziom w grze komputerowej. W dojrzałej inżynierii sytuacja wygląda odwrotnie. Optimum może znajdować się na różnych poziomach dla różnych czynności, a część decyzji być może nigdy nie powinna opuszczać domeny ludzkiej odpowiedzialności. Nie każdy statek potrzebuje autopilota podczas cumowania. Nie każde skrzyżowanie powinno działać bez możliwości ręcznej interwencji. Nie każda decyzja, którą można zautomatyzować, powinna zostać zautomatyzowana.

Tak rozumiany agent-centric SDLC przestaje być linią produkcyjną i zaczyna przypominać system śluz. Agent otrzymuje przestrzeń działania, ale przejście do kolejnej strefy wymaga spełnienia konkretnych warunków: testów, dowodów, logów, zgodności z planem albo akceptacji człowieka. Im bliżej systemu produkcyjnego i im większy potencjalny koszt błędu, tym cięższa staje się śluza.

Nie jest to biurokratyczny balast nakładany na technologię, lecz cena umożliwiająca wykorzystanie jej prędkości bez oddawania jej niekontrolowanej władzy nad skutkami.

W następnym poziomie tej architektury pojawia się jednak trudniejsze pytanie. Skoro agent otrzymuje narzędzia, pamięć, dostęp do repozytoriów, możliwość modyfikowania plików, wykonywania poleceń i wywoływania usług, przestaje być wyłącznie generatorem kodu. Staje się podmiotem operacyjnym w technicznym sensie tego słowa, choć nie staje się podmiotem odpowiedzialności prawnej. To właśnie ta asymetria otwiera problem Excessive Agency, prompt injection, zatruwania kontekstu i niekontrolowanych uprawnień, wymuszając stworzenie twardych guardrails. Wtedy rozmowa o produktywności musi ustąpić rozmowie o władzy: kto może co zrobić, na jakiej podstawie, w czyim imieniu, z jakim śladem audytowym i kto ma prawo natychmiast powiedzieć maszynie „stop". I właśnie od tego miejsca należy rozpocząć analizę ładu zarządczego AI.

Agentowy SDLC bez governance byłby bowiem nie tyle nowoczesną fabryką oprogramowania, co zakładem, w którym maszyny nauczyły się same uruchamiać kolejne urządzenia, podczas gdy zarząd wciąż mierzy sukces liczbą wyprodukowanych elementów. Przejście od asystenta generującego kod do agenta wykonującego działania w środowisku informatycznym wprowadza do inżynierii oprogramowania kategorię, którą dotychczas rezerwowano głównie dla prawa, polityki i teorii organizacji: władzę. Nie chodzi oczywiście o władzę w sensie świadomości czy podmiotowości moralnej maszyny, lecz o władzę operacyjną – rozumianą jako realna możliwość zmieniania stanu świata: modyfikowania repozytorium, wykonywania poleceń, uruchamiania procesów, korzystania z API czy ingerowania w bazę danych.

W momencie, w którym model przestaje tylko mówić, a zaczyna działać, bezpieczeństwo AI przestaje być problemem wyłącznie poprawności odpowiedzi, a staje się problemem ustroju uprawnień. To przesunięcie ma konsekwencje głębsze niż kolejna generacja narzędzi deweloperskich. Błąd klasycznego chatbota może sprawić, że człowiek otrzyma nieprawdziwą odpowiedź. Błąd agenta może spowodować, że nieprawdziwe przekonanie zostanie przekształcone w działanie. Halucynacja przestaje być wówczas wyłącznie problemem epistemicznym i staje się zdarzeniem operacyjnym. Model nie musi „wierzyć" w swoją pomyłkę, aby jej konsekwencje były realne.

## Bezpieczeństwo agentowe to zarządzanie uprawnieniami, a nie instrukcjami

Wystarczy, że agent posiada odpowiednie narzędzie, uprawnienie oraz możliwość przejścia od reprezentacji językowej do wykonania. Cassani określa tę architekturę mianem ścisłego ładu zarządczego AI i konstruuje wokół niej gradient autonomii. Na poziomie zerowym maszyna jedynie sugeruje; na pierwszym może wprowadzać proste zmiany, których zatwierdzenie pozostaje domeną człowieka; na drugim działa samodzielnie w wąskich, zdefiniowanych ramach; na trzecim wykonuje zadania w sposób monitorowany, podczas gdy człowiek śledzi przede wszystkim wskaźniki, odchylenia i alerty. Klasyfikacja ta jest wartościowa nie jako matematyczna skala inteligencji, lecz jako metoda wymuszająca pytanie o delegację: co dokładnie oddajemy maszynie wraz z zadaniem? Nie każde narzędzie jest bowiem niewinnym przedłużeniem kompetencji modelu. Narzędzie jest jednocześnie kanałem sprawczości. Dostęp do wyszukiwarki zwiększa możliwość poznania. Dostęp do repozytorium – możliwość ingerencji. Dostęp do API płatniczego – możliwość wywołania skutku ekonomicznego. Z kolei dostęp do poczty, CRM-u, baz klientów czy infrastruktury chmurowej tworzy potencjalny zasięg działania, którego nie sposób oceniać wyłącznie przez pryzmat benchmarków jakości językowej.

Agent nie staje się ryzykowny dlatego, że generuje gorsze zdania. Ryzyko pojawia się wtedy, gdy błąd poznawczy spotyka się z nadmiernym zakresem uprawnień. Właśnie to OWASP określa jako Excessive Agency. W aktualnej klasyfikacji bezpieczeństwa systemów generatywnych problem ten wiąże się z nadmierną funkcjonalnością, uprawnieniami lub autonomią. Szkodliwe działanie może wynikać z halucynacji, błędnego promptu, prompt injection, wadliwego rozszerzenia lub nieprzewidzianej interakcji między agentami.

To fundamentalna zmiana perspektywy. Nie pytamy już tylko, czy model jest „bezpieczny", lecz czy system pozostaje bezpieczny również wtedy, gdy model popełni błąd. Ta druga formuła powinna stać się jedną z naczelnych maksym inżynierii agentowej. Bezpieczeństwo nie może opierać się na założeniu, że model zawsze właściwie zrozumie intencję. W lotnictwie nie projektuje się samolotu pod warunkiem, że pilot nigdy się nie pomyli. W energetyce nie zakłada się, że operator zawsze podejmie właściwą decyzję. W bankowości nie przyjmuje się, że każdy pracownik z szerokimi uprawnieniami okaże się rozsądny. Nowoczesne zarządzanie ryzykiem powstało z odwrotnego założenia: człowiek, urządzenie i procedura mogą zawieść, dlatego system ma ograniczać promień rażenia pojedynczej pomyłki. Agent AI nie zasługuje tutaj ani na pobłażliwość, ani na metafizyczny lęk – zasługuje na tę samą dojrzałość inżynieryjną. Z tego powodu zasada najmniejszych uprawnień (least privilege) zyskuje w świecie agentów nowe znaczenie. Agent powinien posiadać nie tyle

wszystkie narzędzia, które potencjalnie mogą przydać się do realizacji celu, co minimalny zestaw zdolności niezbędny do wykonania konkretnego zadania.

Jeżeli ma przygotować raport, nie potrzebuje prawa do usuwania dokumentów. Jeśli analizuje dane produkcyjne, nie powinien automatycznie móc ich modyfikować. Sugestia migracji bazy nie może oznaczać prawa do jej wykonania. Podobnie w przypadku korespondencji: możliwość wygenerowania tekstu wiadomości to co innego niż uprawnienie do jej samodzielnego wysłania.

Można rzec, że klasyczna informatyka zabezpieczała przede wszystkim granice między użytkownikami, aplikacjami i zasobami. Informatyka agentowa musi dodatkowo zabezpieczyć granicę między intencją a wykonaniem. To właśnie w tej szczelinie powinien pojawić się checkpoint, confirmation gate, policy engine lub człowiek. Nie po to, by ten ostatni mechanicznie zatwierdzał wszystko, lecz dlatego, że pewne klasy działań przekraczają dopuszczalny budżet autonomii. Im większa nieodwracalność skutku, tym silniejszy wymóg niezależnej zgody. Cassani proponuje zatem twarde bariery techniczne: limity modyfikowanych zasobów, zakazy usuwania określonych plików, ograniczenia migracji baz danych, obowiązek ludzkiej akceptacji dla kodu odpowiedzialnego za płatności i autoryzację oraz możliwość szybkiego rollbacku. Koncepcja ta jest znacznie dojrzalsza niż próba „wychowania" agenta dobrym promptem. Prompt jest normą językową; guardrail powinien być normą wykonalną technicznie. Jeśli agentowi nie wolno usuwać tabel produkcyjnych, lepiej odebrać mu prawo wykonania komendy, niż jedynie instruować go w prompcie: „proszę nigdy nie usuwaj tabel produkcyjnych".

Tu objawia się zasadnicza różnica między normą a zabezpieczeniem. Tabliczka „nie dotykać" jest normą. Osłona mechaniczna uniemożliwiająca dotknięcie niebezpiecznego elementu jest zabezpieczeniem. Systemy agentowe wymagają obu, ale w operacjach krytycznych norma nie może zastępować zabezpieczenia. System prompt może instruować model, by nie wykonywał określonych działań, natomiast Role-Based Access Control sprawi, że nawet model ignorujący instrukcję nie będzie miał technicznej możliwości ich realizacji. Najlepszy zakaz to czasem brak klucza do drzwi. Lipiec 2025 roku dostarczył tej zasadzie wyjątkowo sugestywnej ilustracji.

## Błędy agentowe wynikają z luk infrastrukturalnych i braku separacji

Materiał Cassaniego opisuje incydent w Replit jako przykład nadmiernej autonomii agenta, który mimo nałożonych ograniczeń ingerował w dane produkcyjne. Oficjalna relacja firmy potwierdza istotę zdarzenia: Agent usunął dane z bazy aplikacji jednego z użytkowników. Ponieważ środowiska

development i production nie były jeszcze wystarczająco odseparowane, działania podejmowane podczas rozwoju mogły oddziaływać na działającą aplikację. W odpowiedzi Replit rozdzielił bazy development i production, ograniczając Agentowi możliwość ingerencji w bazę produkcyjną oraz wdrażając checkpointy i rollback. W tym miejscu konieczna jest jednak korekta zgodna z przyjętym trybem prawdziwościowym.

W notatce pojawia się informacja o skasowaniu "1206 produkcyjnych baz danych", lecz oficjalny opis Replit nie potwierdza tej liczby. Dlatego nie powinna być ona powtarzana w artykule jako zweryfikowany fakt. Kluczowy jest sam mechanizm wydarzenia: awaria nie wynikała z jakiegoś metafizycznego "buntu maszyny", lecz z połączenia zdolnego agenta, realnego dostępu do zasobów oraz niedostatecznej separacji środowisk. Reakcją firmy nie było więc jedynie stworzenie lepszego promptu, ale zmiana architektury uprawnień. Późniejsze rozwiązania wprost ograniczyły dostęp Agenta do bazy deweloperskiej i oddzieliły ją od produkcyjnej.

Przypadek ten jest cenny, ponieważ pozwala odmitologizować problem. Katastrofa agentowa nie musi przypominać science fiction; może wyglądać jak banalnie znajomy błąd infrastrukturalny: za szerokie uprawnienia, słaba separacja środowisk, brak właściwej bramy decyzyjnej i zbyt duży promień oddziaływania pojedynczego działania. Nowością jest szybkość, z jaką agent przechodzi przez sekwencję takich operacji, oraz fakt, że decyzja wykonawcza powstaje dynamicznie w procesie wnioskowania, a nie w uprzednio napisanym, deterministycznym kodzie. Dlatego architektura bezpieczeństwa proponowana w notatce obejmuje cztery kolejne warstwy: sanityzację wejścia, ograniczenia zachowania, walidację wyjścia oraz niezmienny ślad audytowy. Warto potraktować je nie jako checklistę informatyka, lecz jako cztery odpowiedzi na cztery pytania instytucjonalne. Pierwsze brzmi: czemu agent może wierzyć? Drugie: co może zrobić? Trzecie: co wolno uznać za poprawny rezultat? Czwarte: czy po fakcie potrafimy odtworzyć, co się wydarzyło? Pierwsze pytanie staje się szczególnie dramatyczne w przypadku prompt injection.

Tradycyjne oprogramowanie wyraźnie odróżnia kod od danych. Model językowy otrzymuje natomiast oba te światy często w tej samej reprezentacji językowej. Dokument może zawierać informacje, ale może też zawierać zdanie brzmiące jak instrukcja. Komentarz w repozytorium może opisywać fragment kodu, a jednocześnie nakazywać agentowi wykonanie innego działania. Strona internetowa może być źródłem faktów i równocześnie nośnikiem polecenia skierowanego do systemu, który ją czyta. OWASP uznaje prompt injection za jedno z podstawowych zagrożeń aplikacji opartych na LLM, szczególnie podkreślając wariant pośredni — indirect prompt injection, w którym złośliwa instrukcja znajduje się w zewnętrznym pliku, stronie lub innym źródle przetwarzanym przez model. W systemie agentowym skutkiem nie musi być jedynie zmanipulowana odpowiedź; możliwe jest wywołanie nieautoryzowanych funkcji, ujawnienie

danych albo wykonanie komendy w systemie połączonym z modelem. NIST w analizach bezpieczeństwa agentów zwraca uwagę na ten sam problem: agent czytający wiadomości, strony internetowe czy repozytoria może zostać przechwycony przez instrukcję ukrytą w danych, które miał jedynie analizować.

## Bezpieczeństwo agentowe to architektura nadzoru i weryfikacji, a nie zaufanie do treści

W klasycznej teorii bezpieczeństwa powiedzielibyśmy, że zawiodła granica zaufania. W świecie agentowym granica ta staje się jednak szczególnie subtelna. Agent musi analizować treść, aby wykonać zadanie, a jednocześnie nie może traktować każdej napotkanej informacji jak instrukcji nadrzędnej. Prowadzi to do jednej z fundamentalnych zasad projektowania systemów agentowych: dane nie mogą automatycznie awansować do rangi rozkazu. Źródło informacji powinno mieć przypisaną klasę zaufania, a możliwość wywołania konkretnego działania musi zależeć od polityki systemu, a nie od tego, jak przekonująco brzmi przeczytane zdanie. Dlatego samo filtrowanie fraz typu „ignore previous instructions" jest niewystarczające. Atak nie musi być oczywisty; może być zakodowany, rozproszony, osadzony w różnych modalnościach lub sformułowany tak, by nie przypominać stereotypowego polecenia.

Aktualne wytyczne OWASP wprost wskazują, że RAG ani fine-tuning nie rozwiązują problemu prompt injection w sposób definitywny. Organizacja musi zatem przyjąć założenie, że treść wejściowa może być wroga, nawet jeśli wygląda jak zwykły dokument. Jest to myśl niemal hobbesowska: bezpieczeństwo systemu nie wynika z wiary w dobrą naturę uczestników, lecz z ustanowienia reguł ograniczających skutki ich potencjalnie szkodliwego działania. Agent musi funkcjonować w architekturze, w której nie ufa bezwarunkowo dokumentowi, stronie, innemu agentowi, a nawet własnej poprzedniej odpowiedzi. Zaufanie zostaje zastąpione przez stopniowalną weryfikację. „Trust but verify" w świecie agentowym powinno ustąpić silniejszej formule: verify before privilege. Z tej perspektywy kluczową wartość zyskuje sandboxing. Agent może otrzymać znaczną swobodę działania, pod warunkiem że operuje w środowisku, którego zniszczenie jest tanie i odwracalne. Sandbox nie jest więc więzieniem inteligencji, ale warunkiem jej eksperymentalnej wolności. Im lepiej izolujemy konsekwencje, tym więcej autonomii możemy dopuścić. Pojawia się tu pozorny paradoks: najbezpieczniejsze ograniczenie może umożliwiać największą swobodę. Tor wyścigowy pozwala jechać szybciej właśnie dlatego, że posiada bariery, strefy bezpieczeństwa i jasne reguły.

Drugim filarem jest walidacja rezultatu. Kod wygenerowany przez AI powinien podlegać tej samej logice, która od dekad chroni inżynierię przed ludzkimi błędami: testom, lintowaniu, analizie statycznej, skanowaniu zależności, kontroli sekretów, review i środowiskom stagingowym. Jednak agentowość wprowadza dodatkową potrzebę – porównanie nie tylko rezultatu z testem, ale działania z intencją. Kod może przejść testy, a mimo to realizować funkcję, której nikt nie zamawiał. Może działać poprawnie lokalnie, naruszając jednocześnie zasadę architektoniczną. Może rozwiązać zgłoszenie, wprowadzając przy tym zbędny mechanizm, który poszerza powierzchnię ataku. Właśnie dlatego Plan Wykonania oraz Logbook stają się elementami governance. Dziennik ma dostarczać śladu pozwalającego ustalić nie tylko to, jaki kod powstał, lecz dlaczego agent podjął określone działania i w którym punkcie odszedł od uzgodnionego planu. Materiał Cassaniego opisuje Logbook jako funkcjonalny odpowiednik czarnej skrzynki i narzędzie przeciwdziałania „Dark Knowledge". Można rozwinąć tę analogię: czarna skrzynka nie zapobiega każdej katastrofie, ale sprawia, że zdarzenie to staje się źródłem wiedzy, a nie tylko stratą.

Stąd kolejna zasada – blameless post-mortem. Postuluje ona analizowanie incydentów agentowych nie poprzez poszukiwanie winnego człowieka, lecz poprzez ustalenie, dlaczego konfiguracja, uprawnienia, procesy lub bariery dopuściły do zdarzenia. To lekcja z dojrzałej kultury DevOps i site reliability engineering: jeśli system bezpieczeństwa działa tylko pod warunkiem nieomylności operatora, jest on źle zaprojektowany. W przypadku agentów zasada ta staje się kluczowa, ponieważ część błędów będzie wynikiem interakcji, której żaden człowiek nie zaprogramował linia po linii. Najbardziej humanistycznym elementem tej architektury pozostaje jednak Andon Cord.

Inspiracja pochodzi z tradycji lean manufacturing i Toyota Production System: pracownik powinien mieć prawo zatrzymać linię produkcyjną, gdy dostrzeże problem jakościowy. Cassani przenosi tę zasadę do organizacji agentowej, postulując, aby każdy członek zespołu mógł natychmiast zamrozić lub odłączyć działającego agenta w razie anomalii zagrażającej systemowi, bez obawy o konsekwencje służbowe. To rozwiązanie jest znacznie ważniejsze, niż sugeruje techniczna metafora przycisku awaryjnego. Andon Cord tworzy prawo sprzeciwu wobec automatyzacji. Pracownik nie musi udowodnić katastrofy, by zatrzymać proces; wystarczy uzasadniona obserwacja anomalii. Organizacja sygnalizuje w ten sposób, że bezpieczeństwo ma pierwszeństwo przed chwilową produktywnością. Jest to szczególnie istotne w środowisku, gdzie KPI mogą nieświadomie zachęcać do ignorowania problemów. Jeśli zespół rozliczany jest wyłącznie z velocity, każda pauza będzie wyglądała jak porażka. Jeśli natomiast organizacja mierzy również jakość, odzyskiwalność, bezpieczeństwo i zdolność uczenia się na błędach, zatrzymanie procesu staje się aktem profesjonalizmu. Można wręcz powiedzieć, że dojrzałość AI governance poznaje się nie po tym, jak wiele agentów organizacja potrafi uruchomić, lecz po tym, jak łatwo potrafi je

bezpiecznie zatrzymać. Technologia, której nie umiemy wyłączyć, przestaje być narzędziem – staje się zależnością infrastrukturalną.

## Governance jako techniczna architektura nadzoru

W tym ujęciu rollback, kill switch, checkpoint, separacja środowisk oraz możliwość ręcznej eskalacji stają się odpowiednikami konstytucyjnych mechanizmów hamowania władzy. Monteskiusz nie pisał o agentach AI, lecz jego intuicja pozostaje aktualna: każda koncentracja zdolności działania wymaga przeciwwagi. Prawo europejskie zaczyna materializować tę zmianę, przesuwając akcent z dobrych praktyk na obowiązki instytucjonalne. Warto tu doprecyzować kwestię terminologiczną: AI Act nie jest dyrektywą, lecz rozporządzeniem Unii Europejskiej, co oznacza instrument bezpośrednio obowiązujący w porządku unijnym. Rozporządzenie weszło w życie 1 sierpnia 2024 roku; zakazane praktyki oraz obowiązki związane z AI literacy zaczęły być stosowane 2 lutego 2025 roku, a regulacje dotyczące modeli ogólnego przeznaczenia – 2 sierpnia 2025 roku. Znaczna część pozostałego systemu weszła w życie 2 sierpnia 2026 roku, z uwzględnieniem okresów przejściowych dla części systemów wysokiego ryzyka.

Znaczenie AI Act dla zespołu programistycznego nie sprowadza się do pytania o to, czy dany agent jest systemem wysokiego ryzyka. Kluczowa jest zmiana kultury odpowiedzialności, którą regulacja ta wzmacnia. Niezbędna staje się wiedza o tym, jaki system jest używany, w jakim celu, na jakich danych, przez kogo, pod czyim nadzorem i z jakim ryzykiem. Nawet jeśli konkretny agent programistyczny nie podlega najbardziej rygorystycznemu reżimowi prawnemu, przedsiębiorstwo działające w Europie nie może budować infrastruktury AI według zasady „uruchomimy, zobaczymy, potem napiszemy politykę". Governance nie powinien przy tym zamienić się w korporacyjny teatr dokumentacyjny. Najgorszą odpowiedzią na realne ryzyko technologiczne jest produkcja martwej instrukcji, którą wszyscy podpiszą, a nikt nie będzie stosował. Jeśli zasada bezpieczeństwa nie zostanie zaszyta w workflow, szybko przegra z wygodą. Dlatego istotny jest proponowany w materiale RAUC (Responsible AI Usage Checklist), który ma trafiać bezpośrednio do pre-commitów, Pull Requestów i codziennych rytuałów pracy, budując „pamięć mięśniową" odpowiedzialnego korzystania z AI. Dobra norma organizacyjna staje się bowiem skuteczna dopiero wtedy, gdy pojawia się w chwili podejmowania decyzji, a nie pół roku później podczas audytu.

Najważniejsza przemiana polega na tym, że AI governance schodzi z sali zarządu do architektury kodu. Polityka staje się permission systemem. Zasada ostrożności – sandboxem. Odpowiedzialność

– audit logiem. Rozdział obowiązków – separacją ról. Kontrola – checkpointem. Prawo sprzeciwu zaś Andon Cordem.

Pamięć instytucjonalna staje się Logbookiem. To właśnie w punkcie, w którym wartości organizacyjne zostają przełożone na mechanizmy techniczne, kończy się deklaracja, a zaczyna rzeczywisty ład. Dojrzały system agentowy nie powinien dążyć do stworzenia maszyny, której można ufać bezwarunkowo; powinien raczej tworzyć środowisko, w którym brak konieczności bezwarunkowego zaufania pozwala bezpiecznie korzystać z jej zdolności. To być może najważniejsza zmiana filozoficzna całej adopcji. Nie budujemy nowego cyfrowego pracownika, któremu powierzamy organizację tylko dlatego, że jest genialny. Budujemy system podziału kompetencji, w którym człowiek i maszyna mogą popełniać błędy, lecz żadna pojedyncza pomyłka nie zyskuje władzy nad całością.

Dopiero z tego punktu można przejść do kolejnego problemu. Gdy guardrails wyznaczą granice działania, pojawia się pytanie bardziej subtelne: jak ludzie i agenci mają wspólnie myśleć? Nie wystarczy przydzielić uprawnień; trzeba zorganizować przepływ intencji, dialogu, walidacji i integracji, przechodząc od AI governance do architektury współpracy. W tej przestrzeni pojawiają się ECF i sekwencja IDVI, nowe role Context Engineera, AI Code Reviewera i Tooling Specialist, problem „ciemnej wiedzy", systemy wieloagentowe oraz pytanie o kształt zespołu przyszłości: czy będzie on grupą ludzi wyposażonych w narzędzia, czy raczej organizacją inteligencji, w której ludzie i maszyny otrzymują różne kompetencje, zakresy odpowiedzialności i prawa do podejmowania decyzji. Governance odpowiada na pytanie, co agentowi wolno zrobić; kolejny poziom dojrzałości musi odpowiedzieć na pytanie znacznie trudniejsze: jak człowiek i maszyna mają wspólnie dochodzić do decyzji, skoro nie myślą w ten sam sposób, nie posiadają tych samych ograniczeń i nie ponoszą tej samej odpowiedzialności?

Samo połączenie kompetentnego programisty z kompetentnym modelem nie tworzy automatycznie kompetentnego zespołu. Historia organizacji wielokrotnie dowiodła, że suma inteligentnych jednostek nie jest tożsama z inteligencją zbiorową. Może powstać orkiestra albo kakofonia; parlament albo tłum; laboratorium albo komitet produkujący eleganckie uzasadnienia wcześniej popełnionych błędów. Problem agentów AI odtwarza tę starą prawdę w nowym medium.

# Framework ECF przesuwa punkt ciężkości z jakości promptu na jakość procesu współpracy

Cassani proponuje Effective Collaboration Framework, którego rdzeniem jest sekwencja IDVI: Intent, Dialogue, Validation, Integration. Jest to czteroetapowy cykl pracy hybrydowej, mający na celu ograniczenie bezkrytycznego przejmowania wyników generowanych przez AI oraz zapobieganie zamykaniu wiedzy w pojedynczych interakcjach człowieka z modelem.

Nie jest to obecnie uznany standard naukowy, jak protokół sieciowy czy norma ISO; należy go traktować jako propozycję organizacyjną Cassaniego. Jego wartość polega jednak na przesunięciu uwagi z samej jakości odpowiedzi AI na jakość procesu współpracy – procesu, w którym odpowiedź powstaje, zostaje podważona, zweryfikowana i dopiero później włączona do systemu. Pierwszym elementem jest Intent, czyli intencja. W tradycyjnym programowaniu wymaganie trafiało do dewelopera w postaci ticketu, user story, specyfikacji lub rozmowy z product ownerem, a człowiek dokonywał milczącej translacji języka biznesowego na techniczny. Lata doświadczenia pozwalały mu wyczuć, że sformułowanie „użytkownik powinien szybko odzyskać dostęp" ukrywa kwestie autoryzacji, bezpieczeństwa, wygasania tokenów, audytu i UX.

Agent językowy również dokonuje translacji, lecz opiera się na otrzymanym kontekście. Jeśli intencja jest nieprecyzyjna, maszyna niekoniecznie zareaguje protestem; może zamiast tego zbudować bardzo sprawne rozwiązanie niewłaściwego problemu. Dlatego Intent-Driven Development nie jest jedynie elegancką nazwą dla lepszego promptowania. Jest próbą odzyskania prymatu celu nad wykonaniem. Zanim człowiek zastanowi się, co ma napisać agent, powinien wiedzieć, jaki stan rzeczy chce osiągnąć, jakie ograniczenia muszą zostać zachowane i po czym pozna sukces. Dojrzalsza AI paradoksalnie zwiększa zatem znaczenie precyzji ludzkiej myśli. Maszyna potrafi tanio wyprodukować odpowiedź na nieprecyzyjne pytanie, co sprawia, że koszt takiego pytania w rzeczywistości rośnie. Zasada „garbage in, garbage out" nie znika wraz z generatywną AI.

Otrzymuje turboładowanie. Drugi etap, Dialogue, jest jeszcze bardziej interesujący, gdyż przeczy rozpowszechnionemu mitowi „perfekcyjnego promptu". W logice ECF dobra współpraca nie polega na sformułowaniu jednego magicznego polecenia, po którym maszyna objawi właściwe rozwiązanie. Dialog to iteracyjne doprecyzowanie problemu, badanie alternatyw, ujawnianie założeń, konfrontowanie modelu z kontrprzykładami i pozwalanie mu wskazywać luki w specyfikacji.

# Od promptowania do inżynierii kontekstu i walidacji

Człowiek przestaje być autorem komendy, a staje się prowadzącym negocjację znaczenia. To różnica podobna do tej między wydaniem rozkazu a prowadzeniem seminarium sokratejskiego: celem nie jest uzyskanie natychmiastowej odpowiedzi, lecz ujawnienie, co właściwie trzeba wiedzieć, aby odpowiedź miała sens. Nie oznacza to oczywiście, że model staje się równoprawnym partnerem epistemicznym w ludzkim znaczeniu tego terminu. Nie ponosi on odpowiedzialności za przekonania, nie ma interesu w prawdzie w takim sensie jak naukowiec i może z jednakową stylistyczną pewnością przedstawić trafną diagnozę oraz subtelną konfabulację. Dialog jest więc wartościowy pod warunkiem zachowania asymetrii: AI rozszerza przestrzeń hipotez, człowiek zaś nie abdykuje z obowiązku osądu. Najgorszą wersją współpracy nie jest maszyna popełniająca błąd, lecz człowiek, który rezygnuje z własnego sceptycyzmu dlatego, że błąd został wypowiedziany językiem kompetencji.

Trzeci element IDVI, Validation, stanowi z tego względu oś całego modelu. Generatywna sztuczna inteligencja uczyniła tworzenie hipotez tanim, ale nie sprawiła, by równie tanie było dowodzenie ich prawdziwości. W programowaniu wygenerowanie dziesięciu wariantów implementacji może kosztować mniej niż rzetelne sprawdzenie jednego. Powstaje zatem asymetria produkcji i falsyfikacji. Karl Popper zapewne rozpoznałby tutaj stary problem nauki w nowym kostiumie: wartość hipotezy nie rośnie wraz z liczbą zdań, które potrafimy wygenerować w jej obronie, lecz od jakości prób, którym pozwalamy ją poddać.

Walidacja musi więc być projektowana jako proces niezależny od generacji. Test napisany przez ten sam model, który stworzył kod, może odziedziczyć to samo błędne założenie. Agent dokonujący review własnego rozwiązania może pozostać przywiązany do trajektorii, którą sam zbudował. Stąd w materiale Cassaniego pojawia się idea potoków walidacyjnych łączących klasyczne testy, lintery i narzędzia CI/CD z niezależnymi agentami kontrolnymi. Warto rozwinąć tę zasadę jako separację epistemicznych interesów: jeden element systemu proponuje, drugi próbuje obalić; jeden optymalizuje rozwiązanie, drugi szuka przypadku brzegowego; jeden pyta „jak to zrobić?”, a drugi „dlaczego to może się nie udać?”.

Dopiero czwarta faza, Integration, pozwala wynikowi stać się częścią rzeczywistego systemu. Integracja jest czymś więcej niż merge'em – obejmuje zgodność z architekturą, stylem zespołu, wymaganiami bezpieczeństwa, dokumentacją, obserwowalnością, modelem danych i wiedzą organizacji. Można wygenerować fragment kodu doskonały lokalnie, który globalnie pogarsza system; napisać znakomitą funkcję, która powiela już istniejącą lub poprawić wydajność jednego komponentu, przerzucając koszt na inny. Można wreszcie stworzyć rozwiązanie tak pomysłwe, że za

pół roku nikt nie będzie chciał go dotknąć. Integracja jest więc momentem, w którym rezultat inteligencji lokalnej zostaje skonfrontowany z racjonalnością całego organizmu.

W tym punkcie odsłania się jedna z najważniejszych transformacji profesji informatycznych: rośnie znaczenie inżynierii kontekstu. Materiał źródłowy nadaje Context Engineerowi odpowiedzialność za firmową ontologię, pamięć trwałą i przepływ informacji do systemów AI, a analitykom biznesowym przypisuje coraz większą rolę w formalizowaniu reguł domenowych, pojęć i słowników, z których korzystają agenci. Jest to propozycja organizacyjna, lecz kierunek ten ma silne oparcie w aktualnej praktyce technicznej. Anthropic w 2025 roku opisał context engineering jako rozwinięcie wcześniejszego prompt engineeringu: problemem przestaje być wyłącznie znalezienie właściwych słów instrukcji, a staje się nim kuratorowanie całego zestawu informacji dostępnych modelowi podczas kolejnych kroków działania – systemowych instrukcji, narzędzi, historii, danych zewnętrznych i pamięci.

Prompt engineering koncentrował się na zdaniu; context engineering koncentruje się na świecie, który widzi maszyna. Jeśli agent ma podejmować decyzję architektoniczną, nie wystarczy poinformować go: „stosuj dobre praktyki". Musi wiedzieć, jakie są praktyki tej konkretnej organizacji, które moduły są legacy, które interfejsy posiadają gwarancję kompatybilności, jakie zależności są zabronione i jaka terminologia ma określone znaczenie biznesowe. Musi rozumieć, które decyzje zostały wcześniej odrzucone i dlaczego. Innymi słowy, agent potrzebuje nie encyklopedii wszystkiego, lecz właściwie zrekonstruowanej pamięci instytucjonalnej.

## Od bibliotek kontekstu do ryzyka utraty zrozumienia systemu

Badania nad rzeczywistymi projektami open source zaczynają już uchwycać ten proces. Analiza z 2025 roku dotycząca plików konfiguracyjnych przekazujących kontekst agentom programistycznym, obejmująca setki repozytoriów, wskazuje, że projekty zapisują dla maszyn informacje o strukturze systemu, budowaniu, testowaniu, stylu kodowania oraz regułach działania. Jednocześnie nie wykształciła się jeszcze jedna, stabilna struktura takich materiałów. Jest to cenna obserwacja, gdyż obrazuje powstawanie nowego rodzaju dokumentacji – przeznaczonej nie tylko dla człowieka, ale i dla syntetycznego uczestnika procesu wytwórczego. Materiał Cassaniego trafnie przechodzi więc od dawnych bibliotek promptów ku Context Libraries: wersjonowanym repozytoriom zawierającym instrukcje, formaty planów, logbooki, ograniczenia projektowe i wzorce właściwe dla określonych zadań. To istotny etap profesjonalizacji; wiedza o tym, „jak dobrze

pracować z AI", przestaje być prywatną sztuczką utalentowanego programisty, a staje się aktywem organizacji.

Firma, która nie instytucjonalizuje skutecznych kontekstów, znajduje się w sytuacji podobnej do przedsiębiorstwa, w którym każdy pracownik przechowuje procedury wyłącznie we własnej pamięci. Ta zmiana prowadzi jednak prosto do problemu Dark Knowledge. Cassani ostrzega, że łatwość generowania kodu może doprowadzić do stanu, w którym zespół stopniowo przestaje rozumieć system, który formalnie utrzymuje. Formułuje przy tym niezwykle użyteczną zasadę: nie należy mergować kodu, którego człowiek nie potrafi wyjaśnić i utrzymać. Być może właśnie ta maksyma zasługuje na status jednej z naczelnych zasad profesjonalizmu ery AI.

W tradycyjnym zespole wiedza ukryta była problemem dobrze znanym. „Tylko Marek wie, jak działa billing" lub „nie dotykamy tego modułu, odkąd odszedł jego autor" to klasyczne przykłady ryzyka kluczowego człowieka. Automatyzacja agentowa może pozornie rozwiązać ten problem, gdyż maszyna potrafi analizować ogromne repozytoria i rekonstruować zależności. Jednocześnie może stworzyć znacznie poważniejszą odmianę tego zagrożenia: system, którego nie rozumie już żaden człowiek, ponieważ każdą kolejną warstwę komplikacji naprawia inna maszyna. Ryzyko key person może zmienić się w ryzyko no person. Można to określić mianem paradoksu kompetencji zastępczej. Im lepiej AI radzi sobie z zadaniami, których człowiek nie chce wykonywać, tym mniej okazji ma on, by nabywać kompetencje niezbędne do kontroli AI w przypadkach nietypowych. Automatyzujemy debugging juniorowi, a następnie odkrywamy, że junior nie nauczył się debugować. Automatyzujemy pisanie testów, po czym przyszły senior potrafi uruchomić pakiet testowy, lecz gorzej rozumie, jaki test należało w ogóle zaprojektować. Automatyzujemy analizę legacy code, aż w końcu organizacja posiada dokumentację generowaną przez maszynę dla kodu generowanego przez maszynę i recenzowanego przez kolejną maszynę. Wszystko jest opisane – a coraz mniej ludzi naprawdę wie, co ten opis oznacza.

Dlatego materiał źródłowy kładzie nacisk na Active Mentorship. Senior nie powinien już tylko pokazywać juniorowi gotowych rozwiązań, lecz odsłaniać proces oceny rezultatu AI: dlaczego konkretny prompt był niewystarczający, gdzie model wykonał nieuprawniony skok logiczny, jak rozpoznać pozornie elegancką abstrakcję i kiedy wynik należy odrzucić mimo przechodzących testów. To fundamentalna zmiana pedagogiczna. W świecie obfitości odpowiedzi edukacja musi mniej koncentrować się na samym wytwarzaniu treści, a bardziej na kalibracji osądu. Wyłania się zatem rola AI Code Reviewera. Według Cassaniego nie jest to po prostu dawny reviewer z nowym narzędziem, lecz specjalista świadomy charakterystycznych błędów kodu generowanego przez LLM: halucynowanych zależności, nieefektywnych algorytmów, pozornej poprawności zabezpieczeń, subtelnych podatności czy nadmiernego komplikowania architektury. Podobnie AI

Tooling Specialist lub Agent Orchestrator ma zarządzać nie pojedynczym promptem, lecz całym portfelem modeli, agentów, kosztów, integracji i przepływów pracy. W ten sposób dawna specjalizacja „programisty konkretnego języka" jest stopniowo uzupełniana przez rolę projektanta środowiska, w którym różne inteligencje wykonują pracę.

Nie oznacza to jednak, że klasyczne kompetencje tracą na wartości. Przeciwnie – zaczynają pełnić funkcję warstwy kontrolnej. Kto nie rozumie SQL, nie będzie idealnym arbitrem jakości zapytań wygenerowanych przez agenta. Kto nie zna mechanizmów autoryzacji, nie rozpozna subtelnej luki w „gotowym" module bezpieczeństwa. Kto nigdy nie zgłębił struktur danych, może otrzymać od modelu algorytm wyglądający imponująco, lecz nie posiądzie aparatu pozwalającego dostrzec jego koszt obliczeniowy. Automatyzacja nie likwiduje więc wiedzy podstawowej; czyni jedynie szczególnie niebezpieczną sytuację, w której organizacja zachowuje władzę zatwierdzania bez zachowania kompetencji rozumienia.

## Wieloagentowość to wybór architektury, a nie automatyczny wzrost wydajności

Na tym tle pojawiają się systemy wieloagentowe. Intuicja stojąca za nimi jest kusząca: skoro zespoły ludzkie opierają się na specjalizacji, dlaczego nie zastosować tego samego w pracy maszynowej? Jeden agent mógłby pełnić rolę architekta, drugi kodera, trzeci QA, czwarty eksperta od bezpieczeństwa, a piąty krytyka. Wczesny ChatDev z 2023 roku eksperymentował właśnie z takim modelem, wykorzystując komunikujące się agenty w kolejnych fazach projektowania, programowania i testowania oprogramowania. Autorzy projektu przedstawili język naturalny jako wspólną warstwę koordynacji między wyspecjalizowanymi rolami. To dobry przykład narodzin idei, lecz nie dowód na to, że samo pomnożenie agentów rozwiązuje problem niezawodności.

W tym miejscu konieczna jest korekta entuzjazmu. Wiele źródeł opisuje systemy wieloagentowe w sposób nadmiernie optymistyczny, przywołując m.in. rzekomą debatę Cognition-Anthropic z czerwca 2024 roku. Jednakże kwerenda w źródłach pierwotnych nie potwierdziła istnienia tak jednoznacznie zdefiniowanego publicznego wydarzenia w opisanej formie.

Znacznie lepiej udokumentowany jest natomiast sam spór merytoryczny: wieloagentowość może zwiększać wydajność tam, gdzie zadanie rzeczywiście daje się rozdzielić, lecz jednocześnie generuje nowe problemy związane z koordynacją, kosztami i propagacją błędów. Pouczające są w tym kontekście doświadczenia Anthropic z systemem wieloagentowym do researchu. Firma opisała architekturę, w której agent główny deleguje niezależne kierunki poszukiwań subagentom

dysponującym osobnymi oknami kontekstowymi. W wewnętrznym benchmarku system ten przewyższył wariant jednoagentowy w określonych zadaniach badawczych, lecz wiązało się to z radykalnie wyższym zużyciem tokenów. Anthropic podkreśla przy tym, że rozwiązanie najlepiej sprawdza się przy zadaniach dających się silnie zrównoleglić; wiele problemów programistycznych posiada jednak więcej zależności i mniej niezależnych ścieżek niż research.

To niezwykle cenna lekcja: multi-agent nie jest wyższym poziomem ewolucji single-agent; jest architekturą właściwą dla określonej geometrii problemu. Jeżeli zadanie można rozbić na dziesięć niezależnych wyszukiwań, dziesięciu agentów może rzeczywiście pracować równolegle. Jeśli jednak każde kolejne działanie zależy od subtelnego rezultatu poprzedniego, pomnożenie wykonawców jedynie mnoży koszty komunikacji. Organizacja ludzka zna to zjawisko od dawna: dziewięć kobiet nie urodzi dziecka w miesiąc.

Nie wszystkie procesy są skalowalne poprzez dodawanie równoległych wykonawców. Frederick Brooks sformułował dla inżynierii oprogramowania podobną intuicję w słynnej książce „The Mythical Man-Month”: komunikacja staje się kosztem rosnącym wraz z liczebnością zespołu. Agenci nie uzyskali od natury zwolnienia z tej logiki.

## Wieloagentowość nie usuwa zarządzania, lecz zmienia jego naturę

Eksperyment Anthropic z lutego 2026 roku, w którym szesnaście równoległych agentów budowało kompilator C w Rust zdolny ostatecznie kompilować jądro Linuksa na kilku architekturach, pokazuje zarówno potencjał, jak i ograniczenia tej metody. Projekt wymagał prawie dwóch tysięcy sesji agentowych, około stu tysięcy linii kodu oraz wydatku rzędu 20 tysięcy dolarów na API. Znaczną część inżynierii poświęcono nie samemu "myśleniu modeli", lecz harnessowi, blokadom zadań, testom, izolowanym repozytoriom i mechanizmom zapobiegającym wykonywaniu tej samej pracy przez wiele instancji. To eksperyment jednego laboratorium i nie powinien być traktowany jako uniwersalna miara produktywności, lecz doskonale pokazuje, że zespół agentów potrzebuje instytucji niemal tak samo jak zespół ludzi. Najpierw trzeba ustalić role, a potem granice odpowiedzialności.

Konieczny jest mechanizm rezerwowania pracy, aby dwie jednostki nie naprawiały tego samego problemu. Trzeba zdecydować, kto integruje rozwiązania, jak rozstrzygać konflikty, kiedy jedno działanie staje się źródłem prawdy dla kolejnych i jakie testy są wspólne. Innymi słowy, wieloagentowość nie usuwa zarządzania; ona produkuje popyt na zarządzanie maszynami.

Najnowsze badania dodatkowo studzą przekonanie, że koordynacja pojawi się spontanicznie wraz ze wzrostem możliwości modeli. Benchmark opublikowany w 2026 roku, badający otwartą wieloagentową koordynację w długich zadaniach, wskazał znaczącą różnicę między kompetencją pojedynczego agenta a kompetencją koordynacyjną zespołu. Modele potrafiące dobrze radzić sobie z samym zadaniem niekoniecznie równie sprawnie rozdzielały role, komunikowały się i utrzymywały wspólny plan; w badanym środowisku komunikacja okazała się jednym z głównych czynników wpływających na zdolność współdziałania. Jest to rezultat zgodny z intuicją teorii organizacji: inteligencja zbiorowa wymaga protokołu, nie tylko inteligentnych uczestników.

Dlatego rola orkiestratora staje się centralna. Orkiestrator nie powinien być rozumiany jako "najmądrzejszy agent", który wszystko wie. Jego funkcją jest właściwe dzielenie problemu, przypisywanie kompetencji, przekazywanie minimalnego koniecznego kontekstu, pilnowanie zależności oraz integrowanie wyników. Anthropic w praktyce własnego systemu researchowego zaobserwował, że nieprecyzyjne delegowanie prowadziło do powielania pracy, pozostawiania luk albo eksplorowania różnych znaczeń tego samego zadania przez niezależne subagenty. Problem delegacji powraca więc niemal w tej samej formie, w jakiej od wieków występuje w administracji i zarządzaniu ludźmi: źle sformułowane zadanie jest skalowalnym źródłem źle wykonanej pracy.

W tym miejscu ECF i IDVI nabierają nowego znaczenia. Intent nie musi już przepływać jedynie od człowieka do pojedynczego modelu; musi zostać zachowany w kolejnych delegacjach pomiędzy agentami. Dialogue obejmuje komunikację poziomą i pionową. Validation musi sprawdzać nie tylko produkty pracy, ale również poprawność handoffów. Integration musi składać rezultaty powstałe w odrębnych kontekstach w jedną spójną całość.

Im więcej agentów dodajemy, tym większą część problemu stanowi utrzymanie semantycznej ciągłości celu. Można nazwać to problemem głuchego telefonu automatyzacji. Pierwszy agent otrzymuje złożoną intencję biznesową, redukuje ją do zadania dla drugiego, ten do technicznego podproblemu dla trzeciego, a czwarty otrzymuje już niezwykle precyzyjne polecenie – które realizuje perfekcyjnie, choć niewiele ma ono wspólnego z pierwotnym celem. Każdy lokalny krok może wyglądać racjonalnie; katastrofa pojawia się dopiero na poziomie całości.

Dlatego nowoczesne architektury eksperymentują z trwałymi artefaktami, pamięcią zewnętrzną i bezpośrednim zapisem rezultatów zamiast wielokrotnego przepisywania ich przez kolejne warstwy agentów. Anthropic opisuje podobną technikę jako sposób ograniczania utraty informacji w wieloetapowych handoffach.

# Od rzemiosła pisania do eksperckości opartej na osądzie

Inżynieria kontekstu staje się zatem odpowiednikiem zarządzania pamięcią organizacyjną. Najlepszy kontekst nie jest tym największym, lecz najmniejszym zbiorem informacji o najwyższej wartości dla aktualnej decyzji. Podobną zasadę formułuje Anthropic: efektywny context engineering powinien maksymalizować wartość sygnału przy ograniczonym budżecie uwagi modelu, wykorzystując m.in. pobieranie informacji „just in time", kompresję, notatki strukturalne oraz podział pracy między subagentów. Nowsze badania nad agentami SWE wskazują na podobny kierunek: długie trajektorie bez aktywnego zarządzania kontekstem prowadzą do jego eksplozji, semantycznego dryfu i spadku jakości rozumowania. Dlatego pojawiają się metody traktujące kompresję i porządkowanie pamięci jako jawne działanie samego agenta.

Prowadzi to do jednej z najbardziej przewrotnych lekcji rewolucji AI: przez lata marzyliśmy o modelach z coraz większym oknem kontekstowym, jak gdyby doskonała inteligencja miała powstać przez wrzucenie całej biblioteki na biurko. Tymczasem praktyka pokazuje, że uwaga pozostaje zasobem nawet w przypadku maszyny. Herbert Simon pisał, że bogactwo informacji rodzi ubóstwo uwagi.

W świecie agentów maksyma ta zyskała wymiar techniczny. System mający dostęp do wszystkiego może działać gorzej niż ten, który otrzymuje dokładnie to, czego potrzebuje w odpowiednim momencie. Przyszły zespół technologiczny może więc przypominać nie tyle grupę programistów siedzących obok siebie przy klawiaturach, co rynek wyspecjalizowanych inteligencji zarządzany przez ludzi odpowiedzialnych za intencję, kontekst, walidację i integrację. Jedna warstwa będzie formułowała cele biznesowe, inna utrzymywała ontologię domenową, kolejna konfigurowała agentów, a jeszcze następna badała bezpieczeństwo. Człowiek z wiedzą architektoniczną zdecyduje natomiast, które rezultaty otrzymają prawo wejścia do systemu. Materiał Cassaniego nazywa taki zespół „hybrydowym organizmem", w którym programista staje się superwizorem i dyrygentem systemów poznawczych.

Metafora ta jest trafna pod jednym zastrzeżeniem: organizm pozostaje zdrowy tylko wtedy, gdy zachowuje własny układ nerwowy. Organizacja, która zautomatyzuje wykonanie, tracąc jednocześnie zdolność rozumienia wykonywanej pracy, nie stanie się inteligentniejsza – stanie się zależna. Dlatego najbardziej wartościowym pracownikiem ery agentowej może nie być ten, kto potrafi wydobyć z AI najwięcej kodu, lecz ten, który najlepiej rozpozna, jaki kod nie powinien powstać, jakiej informacji modelowi brakuje, którego wyniku nie wolno zaakceptować i kiedy pięciu zgodnych ze sobą agentów wciąż może się mylić. Stary ideał eksperta opartego na pamięci ustępuje

miejsca ekspertowi opartemu na osądzie. Wiedza encyklopedyczna tanieje; zdolność budowania hierarchii ważności, rozpoznawania wyjątków, rekonstruowania konsekwencji i stawiania właściwego veto pozostaje zasobem rzadkim.

W ten sposób zmiana technologiczna dotyka samej konstrukcji kariery zawodowej. Jeśli junior coraz rzadziej rozpoczyna od mozolnego pisania boilerplate'u, QA rzadziej ręcznie produkuje setki prostych testów, analityk przestaje być jedynie autorem wymagań, a senior nadzoruje agentów zamiast wykonywać wszystkie czynności osobiście, trzeba zapytać nie o przyszłość stanowisk, lecz o proces stawania się ekspertem. Dotychczas wiedza seniora wyrastała przecież z tysięcy małych błędów popełnianych w czasach junioratu. Jeśli maszyna zabierze młodemu człowiekowi błędy, może przypadkiem odebrać mu część edukacji. W tym miejscu technologiczna opowieść o AI spotyka się z antropologią pracy.

Fachowości nie nabywa się wyłącznie przez otrzymywanie poprawnych wyników. Rodzi się ona także z frustracji, nieudanych prób, debugowania, odpowiedzialności za konsekwencje oraz stopniowego budowania intuicji, której nie da się zamknąć w instrukcji. Jeśli chcemy zbudować prawdziwie hybrydową organizację inteligencji, nie możemy automatyzować wszystkiego, co nudne, tylko dlatego, że potrafimy. Część nudnych zadań jest bowiem siłownią poznawczą, na której powstają kompetencje niezbędne później do nadzorowania maszyn. Kolejna granica artykułu prowadzi zatem ku człowiekowi.

Po architekturze agentów, gradientach autonomii, governance i organizacji współpracy należy zbadać, co rewolucja agentowa robi z zawodem, zespołem, hierarchią kompetencji i przywództwem. Tam pojawi się najtrudniejszy paradoks adopcji: przedsiębiorstwo chce dzięki AI eliminować pracę wymagającą najmniej doświadczenia, a jednocześnie potrzebuje ludzi coraz bardziej doświadczonych, by skutecznie kontrolować rezultaty. Jeśli nie zbuduje nowej drabiny uczenia się, może odkryć, że posiada dziesiątki znakomitych agentów, garstkę seniorów i nikogo, kto za kilka lat będzie mógł tych seniorów zastąpić. Kto wychowa seniora?

Kompetencje, deskilling i przywództwo w organizacji hybrydowej. Najbardziej niewygodne pytanie dotyczące automatyzacji programowania nie brzmi: czy sztuczna inteligencja zastąpi programistę? Jest ono zbyt ogólne, zbyt medialne i zbyt łatwo prowadzi do jałowej wojny między technologicznym mesjanizmem a lękiem.

# Kryzys reprodukcji kompetencji i ryzyko deskillingu

Znacznie ciekawsze pytanie brzmi: jeżeli AI przejmie znaczną część pracy, dzięki której początkujący programista stawał się doświadczonym specjalistą, to w jaki sposób organizacja wyprodukuje przyszłego seniora? W tym pytaniu mieści się problem, którego nie da się rozwiązać kolejną licencją, większym modelem ani lepszym promptem. Dotyczy on reprodukcji kompetencji – zdolności organizacji do przekazywania wiedzy z pokolenia na pokolenie. Tradycyjna ścieżka kariery programistycznej była bowiem systemem edukacyjnym, choć rzadko tak ją nazywano. Junior pisał prostszy kod, popełniał błędy i poprawiał je, uczestniczył w code review, obserwował decyzje seniorów oraz otrzymywał zadania nieco przekraczające jego aktualne możliwości. Przeżywał pierwszą awarię produkcyjną, odkrywał różnicę między kodem działającym a utrzymywalnym i stopniowo nabywał intuicję, której nie sposób przyswoić wyłącznie z podręcznika. Programowanie było zawodem poznawanym przez praktykę. Arystotelesowska phronesis – mądrość praktyczna – powstawała nie tylko z wiedzy, lecz z wielokrotnie przeżytego doświadczenia wyboru w warunkach niepewności. Cassani słusznie identyfikuje tutaj niebezpieczeństwo deskillingu.

Automatyzacja boilerplate'u, prostych testów i podstawowego debugowania usuwa właśnie tę część pracy, która dotąd stanowiła pierwszy warsztat młodego inżyniera. Materiał źródłowy proponuje w odpowiedzi Active Mentorship: senior ma nie tylko wyjaśniać gotowy kod, ale ujawniać sposób oceny pracy z AI – pokazywać, dlaczego dana instrukcja jest błędna, jak testować odpowiedź modelu, gdzie szukać ukrytych założeń i kiedy zachować sceptycyzm wobec rozwiązania wyglądającego przekonująco. Jest to jedna z najbardziej dalekowzrocznych tez całej koncepcji, ponieważ przenosi mentoring z poziomu transferu informacji na poziom transferu osądu. Aktualne badania nad kompetencjami programistów wspieranych przez AI wzmacniają ten kierunek, choć nie pozwalają jeszcze na formułowanie prostych praw dotyczących całej profesji. Badanie przeprowadzone wśród 21 profesjonalnych deweloperów intensywnie wykorzystujących AI wskazało, że skuteczność przyszłego programisty wymaga równoczesnego zachowania czterech grup kompetencji: sprawnego używania generatywnej AI, klasycznej wiedzy software engineering, umiejętności z sąsiednich obszarów technicznych oraz kompetencji nietechnicznych.

Autorzy wprost wskazują na potrzebę reskillingu i upskillingu chroniących przed deskillingiem. To ważna korekta wobec popularnej wizji przyszłości, według której wystarczy nauczyć młodego człowieka „dobrze promptować". Prompt bez wiedzy dziedzinowej przypomina świetnie sformułowane pytanie zadawane w języku, którego odpowiedzi nie umiemy ocenić. Powstaje więc paradoks wejścia do zawodu.

# Automatyzacja zadań juniorskich niszczy ścieżkę rozwoju ekspertów

AI w sposób najbardziej naturalny automatyzuje zadania proste, powtarzalne i łatwo sprawdzalne. To właśnie one przez dziesięciolecia stanowiły strefę wejściową dla osób z mniejszym doświadczeniem. Ekonomicznie brzmi to racjonalnie: po co płacić człowiekowi za pracę, którą maszyna wykonuje szybciej?

Organizacyjnie może jednak powstać luka. Jeśli przedsiębiorstwo eliminuje pracę juniorską, nie eliminując zapotrzebowania na seniorów, zachowuje owoce, jednocześnie ścinając sad. Przez kilka lat wszystko może wyglądać znakomicie, gdyż dostępna jest obecna generacja ekspertów. Problem ujawni się później, gdy organizacja odkryje, że kompetencja senioralna nie pojawia się spontanicznie wraz ze zmianą nazwy stanowiska na LinkedInie.

Istotę problemu oddaje rozróżnienie między learning by doing a learning by choosing. W pierwszym modelu człowiek samodzielnie konstruuje rozwiązanie i przechodzi przez proces, który ujawnia mu strukturę problemu. W drugim otrzymuje kilka gotowych propozycji i wybiera najlepszą z nich. Druga forma jest znacznie szybsza, lecz wymaga kompetencji oceny, która historycznie kształtowała się właśnie dzięki tej pierwszej. Dochodzimy więc do osobliwego błędnego koła: AI pozwala ominąć etap praktyki, ale człowiek potrzebuje doświadczenia z tego etapu, aby trafnie oceniać rezultat działania maszyny.

Nie jest to jedynie intuicja filozoficzna. Badanie Microsoft Research, obejmujące 319 pracowników wiedzy i 936 przypadków korzystania z generatywnej AI, wykazało, że większe zaufanie do systemu wiązało się z mniejszym wysiłkiem krytycznego myślenia; natomiast większa pewność własnej wiedzy – z większym zaangażowaniem krytycznym. Jednocześnie sama natura tego myślenia przesuwała się z wytwarzania rozwiązania ku weryfikacji informacji, integracji odpowiedzi i nadzorowaniu zadania. Choć nie wolno automatycznie przenosić wyników badania pracowników wiedzy na każdą sytuację inżynierii oprogramowania, jego struktura współgra z agent-centric SDLC: człowiek rzeczywiście przemieszcza się od wykonania ku task stewardship.

Problem polega na tym, że stewardship wymaga stewarda kompetentnego. Człowiek nie może skutecznie nadzorować procesu tylko dlatego, że interfejs udostępnił mu przycisk „Approve”. Jeśli nie rozumie mechanizmu, nie potrafi przewidzieć klasy błędów i nie wie, które elementy wyniku wymagają podejrzliwości, jego kontrola staje się jedynie rytuałem.

W ten sposób można stworzyć najbardziej niebezpieczny typ organizacji hybrydowej: organizację z formalnie ludzką odpowiedzialnością i faktycznie maszynowym osądem. Dokumenty będą podpisywane przez ludzi, commity zatwierdzane przez ludzi, a decyzje w istocie będą przejmowane z propozycji systemu, ponieważ nikt nie będzie miał czasu ani kompetencji, aby je zakwestionować. W tym kontekście stare powiedzenie rzemieślników – że narzędzie nie czyni rzemieślnika – nabiera nowego znaczenia. AI nie jest jednak zwykłym młotkiem.

To narzędzie, które potrafi zaproponować, gdzie uderzyć, przekonać użytkownika, że wybrało właściwe miejsce, a następnie samo wykonać cios. Dlatego kompetencja człowieka musi przesuwać się z wykonania nie tyle ku bierności, co ku meta-kompetencji: rozumieniu problemu, architektury, jakości dowodu, ograniczeń modelu i konsekwencji decyzji. Cassani mówi w tym kontekście o bifurkacji ścieżek kariery: młody inżynier miałby wcześniej rozwijać zdolności systemowe i architektoniczne, prowadzące ku rolom takim jak Context Engineer, AI Tooling Specialist czy AI Code Reviewer.

Należy jednak potraktować tę propozycję krytycznie. Nie da się po prostu nakazać juniorowi, aby wcześniej został architektem. Architektura nie jest zbiorem terminów, lecz skondensowanym doświadczeniem konsekwencji. Człowiek rozumie wartość prostoty inaczej po przeczytaniu maksymy KISS, a inaczej po dwóch latach utrzymywania systemu, który ktoś wcześniej uczynił zbyt „sprytnym”. Wie, czym jest rollback z dokumentacji, ale zupełnie inaczej rozumie go po pierwszym wdrożeniu, które musiał cofnąć o drugiej nad ranem.

## Utrzymanie ludzkiej rezerwy poznawczej jako element odporności systemu

Mądre przedsiębiorstwo nie powinno zatem usuwać wszystkich zadań edukacyjnych tylko dlatego, że można je zautomatyzować. Część pracy należy zachować lub wręcz sztucznie konstruować jako przestrzeń ćwiczenia fachowości. Piloci szkolą się na symulatorach nie dlatego, że linie lotnicze potrzebują więcej symulowanych lotów, lecz by w razie rzeczywistej potrzeby dysponować ludźmi zdolnymi do prawidłowej reakcji.

Chirurg ćwiczy procedury, których może nie wykonywać codziennie. Wojsko od stuleci ponosi koszt manewrów, choć samo ćwiczenie nie zdobywa żadnego terytorium. Organizacja AI-first musi odkryć analogiczną kategorię: pracę wykonywaną dla utrzymania zdolności, a nie dla natychmiastowego outputu. W praktyce może to oznaczać okresowe sesje programowania bez asystenta, ręczne debugging drills, ćwiczenia z rekonstrukcji przyczyn awarii czy obowiązek

samodzielnego wyjaśnienia kodu przed jego akceptacją. Do tej listy należą także rotacje pomiędzy rolami oraz analiza rozwiązań AI bez wcześniejszej wiedzy o ich autorze. Nie jest to nostalgiczne bronienie rękodzieła przed maszyną, lecz utrzymanie ludzkiej rezerwy poznawczej. Bank nie przechowuje kapitału po to, by leżał bezczynnie; rezerwa istnieje, ponieważ świat nie zawsze zachowuje się zgodnie z prognozą. Kompetencja ludzka jest rezerwą odporności systemu.

Źródło Cassaniego podobnie postuluje utrzymywanie tradycyjnego programowania i rotacji zadań jako odpowiedzi na atrofię ludzkich kompetencji. W jego ujęciu ryzyko to nie jest marginalnym kosztem ubocznym, lecz istotnym elementem macierzy governance. Ta intuicja jest kluczowa, gdyż organizacje mają naturalną tendencję do optymalizacji tego, co łatwo mierzalne. Liczbę ukończonych ticketów można policzyć, velocity wykazać na dashboardzie, a oszczędność czasu przeliczyć na euro. Znacznie trudniej jednak wycenić wiedzę, której pracownik nie utracił dzięki temu, że firma pozostawiła mu przestrzeń do samodzielnego myślenia.

## Lider jako orkiestrator inteligencji i strażnik kompetencji

Właśnie tutaj powinna pojawić się nowa funkcja przywództwa. Cassani określa lidera przyszłości mianem Intelligence Orchestrator. Nie zarządza on już tylko ludźmi, lecz całym układem: ludźmi, agentami, narzędziami, przepływami kontekstu i punktami kontroli. Taka rola wymaga czegoś więcej niż entuzjazmu wobec technologii. Lider musi umieć rozstrzygnąć, gdzie automatyzacja rzeczywiście zwiększa zdolność zespołu, a gdzie jedynie przenosi niewidoczny koszt na review, bezpieczeństwo, szkolenia lub przyszłe utrzymanie. Jego zadaniem nie jest dążenie do jak najwyższego wskaźnika adopcji AI, lecz uzyskanie adopcji właściwej. Różnica ta przypomina relację między lekarzem a sprzedawcą leków: sprzedawca maksymalizuje użycie produktu, lekarz zaś dobiera terapię do konkretnego przypadku.

Lider AI-first powinien myśleć jak ten drugi. Pytanie „ile procent zespołu używa AI?” jest samo w sobie intelektualnie ubogie. Sto procent pracowników może przecież korzystać z niewłaściwego narzędzia w niewłaściwy sposób. Znacznie bardziej istotne są pytania o to, które zadania zostały przyspieszone, gdzie wzrosła jakość, a gdzie liczba poprawek; które kompetencje słabną, czy ludzie rozumieją wygenerowane rozwiązania i czy firma potrafiłaby funkcjonować przez pewien czas po odłączeniu systemu. Materiał Cassaniego proponuje w tym kontekście Team AI Maturity Score, łączący adopcję, wydajność, stabilność jakości oraz opór kulturowy.

Wzoru tego nie należy traktować jako empirycznie zwalidowanego standardu naukowego. Bardziej wartościowa od samej arytmetyki jest ukryta za nim intuicja: dojrzałości nie wolno utożsamiać z intensywnością wykorzystania AI. Organizacja, która używa agentów rzadziej, lecz rozumie ich ograniczenia, mierzy jakość i zachowuje kompetencje ludzi, może być znacznie dojrzalsza od organizacji, która wszędzie nakleiła etykietę "AI-first". Przywództwo agentowe musi zresztą zmierzyć się nie tylko z techniką, ale i z tożsamością zawodową. Cassani wymienia cztery bariery: lęk przed zastąpieniem, strach przed dezaktualizacją kompetencji, utratę poczucia rzemiosła oraz zmęczenie nieustanną zmianą. Każda z nich jest bardziej racjonalna, niż sugerują to narracje o „oporze wobec innowacji". Programista, który przez piętnaście lat budował poczucie własnej wartości na umiejętności pisania dobrego kodu, nie doświadcza neutralnej aktualizacji narzędzi w momencie, gdy słyszy, że coraz większą część kodu ma produkować agent.

Zmienia się symboliczna definicja mistrzostwa. Richard Sennett pisał o rzemieślniku jako człowieku pragnącym wykonać rzecz dobrze dla niej samej. W takim rozumieniu praca nie jest wyłącznie wymianą czasu na wynagrodzenie, lecz źródłem tożsamości. Jeżeli organizacja powie seniorowi: „nie pisz już tyle, zarządzaj maszyną, która pisze", może mu oferować awans funkcjonalny, jednocześnie odbierając część satysfakcji, dzięki której przez lata chciał być dobry w swoim fachu. To właśnie loss of craftsmanship. Nie wystarczy wyjaśnić człowiekowi, że „AI go wzmocni" – trzeba zbudować nowe źródło profesjonalnej dumy. Cassani lokuje je w orkiestracji, architekturze, inżynierii kontekstu, walidacji i mentoringu.

Jest to propozycja przekonująca, ale pod jednym warunkiem: nowe zadania muszą być rzeczywiście uznawane przez organizację za wartościowe. Jeżeli firma mówi seniorowi, że jego najważniejszym zadaniem jest mentoring juniorów, lecz premię nadal wylicza głównie z liczby dostarczonych funkcji, kultura zwycięży nad prezentacją PowerPoint. Jeśli deklaruje, że bezpieczeństwo jest ważniejsze od prędkości, a potem nagradza zespoły wyłącznie za velocity, ludzie szybko nauczą się właściwej odpowiedzi. Jak mawiał Deming: system otrzymuje to, co mierzy, a niekoniecznie to, co deklaruje.

Dlatego reorganizacja w epoce AI wymaga również zmiany systemu awansów. Dotychczas programista budował pozycję poprzez ilość i jakość kodu, znajomość frameworków, rozwiązywanie trudnych bugów i udział w projektowaniu systemu. W organizacji agentowej większą wagę mogą zyskać umiejętności definiowania problemu, projektowania ewaluacji, integrowania wyników wielu agentów, wykrywania błędów w pozornie poprawnych rozwiązaniach, uczenia innych oraz zarządzania niepewnością. Źródło słusznie wskazuje też na rolę HR: profile stanowisk, rekrutacja i system ocen powinny odzwierciedlać jakość bezpiecznej współpracy z AI, a nie jedynie klasyczne mierniki produkcji. Szczególnej uwagi wymaga przy tym pozycja juniora.

Wbrew prostym scenariuszom, AI może być dla niego jednocześnie zagrożeniem dla procesu uczenia się i niezwykłym nauczycielem. Model dostępny w każdej chwili może wyjaśniać składnię, tłumaczyć koncepcje, proponować przykłady, konstruować ćwiczenia i udzielać informacji zwrotnej bez społecznego kosztu zadawania „głupiego pytania". Materiał Cassaniego nazywa AI „zawsze dostępnym mentorem". Problem nie polega więc na tym, że AI zabiera możliwość nauki, lecz na tym, jaki rodzaj uczenia się użytkownik wybiera. Uczeń może zapytać: „napisz mi tę funkcję" albo „nie podawaj rozwiązania; zadawaj pytania, które doprowadzą mnie do niego". Może poprosić o gotowy test lub o wskazanie klas błędów, które sam ma spróbować znaleźć. Może wkleić komunikat wyjątku i otrzymać poprawkę albo przejść przez diagnostykę krok po kroku. Technicznie wszystkie te warianty wykorzystują podobny model; pedagogicznie należą do różnych światów.

Pierwszy maksymalizuje rezultat. Drugi maksymalizuje budowę zdolności do samodzielnego osiągania rezultatu w przyszłości. To właśnie tutaj lider musi zrozumieć, że produktywność krótkoterminowa i zdolność długoterminowa mogą wejść w konflikt. Przedsiębiorstwo potrafi zaoszczędzić godzinę dziś, by zapłacić za nią miesiącem zależności za trzy lata.

## Ryzyko poznawczego odciążenia i granice utraty kompetencji

Badania Microsoftu nad krytycznym myśleniem wskazują, że korzystanie z generatywnej AI przesuwa wysiłek poznawczy w stronę weryfikacji i integracji wyników. Co więcej, większe zaufanie do systemu koreluje z mniejszym zaangażowaniem krytycznym. Raport badawczy Microsoftu z 2026 roku szerzej ujmuje tę samą obawę: przejście od "thinking by doing" do wyboru spośród gotowych rezultatów może osłabiać praktyki podtrzymujące eksperckość, o ile nie zostanie połączone z celowym upskillingiem i projektowaniem narzędzi utrzymujących człowieka poznawczo zaangażowanego. Nie oznacza to jednak, że każda forma cognitive offloading jest szkodliwa. Od tysiącleci przenosimy część procesów poznawczych do narzędzi: pismo odciążyło pamięć, kalkulator zautomatyzował rachunki, a GPS pozwala podróżować bez zapamiętywania map. Historia technologii jest historią rozsądnego zapominania pewnych umiejętności. Nie musimy nostalgicznie bronić każdej kompetencji tylko dlatego, że kiedyś była niezbędna. Kluczowe pytanie brzmi: które umiejętności możemy bezpiecznie utracić, a które stanowią warunek kontrolowania systemu, który je przejmuje?

# Suwerenność poznawcza jako fundament nadzoru nad AI

Nikt rozsądny nie wymaga od programisty pisania kodu maszynowego, by udowodnił swój profesjonalizm. Abstrakcja jest istotą rozwoju informatyki. Jednocześnie architekt systemu musi rozumieć wystarczająco dużo o pamięci, konkurencji, sieciach czy bazach danych, aby przewidywać konsekwencje decyzji podejmowanych na wyższym poziomie abstrakcji. Podobnie AI może stać się kolejną warstwą abstrakcji.

Człowiek nie musi pisać każdej linijki kodu, lecz musi zachować minimalną suwerenność poznawczą: umiejętność rozpoznania, kiedy rezultat jest niebezpieczny, kiedy model nie rozumie problemu i kiedy trzeba zejść poziom niżej. Zamiast pytać wyłącznie o to, czego ludzie muszą nauczyć się, by pracować z AI, zapytajmy: „czego ludzie nie mogą przestać umieć, jeśli mają nad AI zachować realny nadzór?". To drugie pytanie jest bardziej niewygodne, gdyż wymaga czasem świadomej rezygnacji z części natychmiastowych oszczędności.

Empiryczna literatura dotycząca współpracy człowieka z AI pokazuje jednak, że nie musi to być gra o sumie zerowej. Eksperyment terenowy „Cybernetic Teammate" z udziałem 776 profesjonalistów Procter & Gamble wykazał, że jednostki wspierane przez AI osiągały wyniki porównywalne z dwuosobowymi zespołami ludzkimi pracującymi bez AI. System pomagał również przełamywać funkcjonalne silosy między specjalistami technicznymi a komercyjnymi, skracając czas wykonania zadań i poprawiając deklarowane doświadczenia emocjonalne grup korzystających z AI. Choć nie było to badanie zespołów programistycznych i nie należy automatycznie przenosić tych efektów na SDLC, pokazuje ono, że AI może działać nie tylko jako automat wykonawczy, ale jako medium integracji wiedzy.

To nadaje nowy sens formule „cybernetic teammate". Najlepszy współpracownik to nie ten, który wykonuje za nas wszystko. Dobry partner zwiększa nasze możliwości, dostrzega to, co nam umyka, pozwala kwestionować własne założenia i pomaga budować rezultat lepszy niż suma indywidualnych wkładów. Jeśli AI ma funkcjonalnie zasłużyć na miano współpracownika, powinno być projektowane nie tylko po to, by odpowiadać szybciej, ale by pomagać człowiekowi myśleć lepiej. Stąd kluczowe znaczenie psychologicznego bezpieczeństwa.

Materiały Cassaniego sugerują wdrożenie Human-AI Pair Programming, retrospektyw AI oraz stworzenie przestrzeni dla kontrolowanych błędów i normalizację uczenia się nowych metod pracy. Jest to podejście rozsądniejsze niż odgórny nakaz: „od jutra wszyscy używają AI". Technologia zmieniająca fundamenty profesjonalnej tożsamości wymaga eksperymentu, a ten z kolei – prawa

do niepowodzenia. Jeśli pierwsza nieudana próba z agentem stanie się argumentem w ocenie rocznej przeciwko pracownikowi, zespół nauczy się nie eksperymentować lub robić to potajemnie. Obie reakcje są dla organizacji szkodliwe.

Przykład „AI Fridays" z opracowania Cassaniego należy traktować jako element scenariusza wdrożeniowego, a nie uniwersalną receptę. Niemniej sama zasada wydzielenia czasu na eksperyment, dokumentowanie skutecznych praktyk i przekładanie indywidualnych odkryć na wspólną bazę wiedzy jest organizacyjnie wartościowa. Proces ten zakłada, że doświadczenia z kontrolowanych prób mają zasilać firmową pamięć, zamiast pozostawać prywatnym kapitałem kilku entuzjastów.

W ten sposób adopcja przestaje przypominać epidemię przypadkowych narzędzi, a staje się procesem uczenia się instytucji. Lider-agent orchestrator powinien zarządzać nie tylko produktywnością, lecz tempem uczenia się zespołu. Musi wiedzieć, które prompty i wzorce działają, jakie klasy błędów powracają, kto potrafi je rozpoznawać oraz gdzie kompetencje ulegają erozji, a gdzie powstają nowe. Powinien budować mechanizm, w którym każde niepowodzenie agenta nie tylko zostaje naprawione, ale zwiększa wiedzę całej organizacji. Dopiero wtedy AI tworzy efekt kumulacyjny: nie tylko wykonuje zadania, lecz pomaga instytucji stawać się lepszą w projektowaniu kolejnych wyzwań.

W tym miejscu klasyczna teoria organizacyjnego uczenia się spotyka się z agentową inżynierią. Chris Argyris rozróżniał single-loop learning, w którym organizacja koryguje błąd bez kwestionowania podstawowych reguł, oraz double-loop learning, w którym bada same założenia prowadzące do błędu. Agent może pomóc na obu poziomach: poprawić funkcję lub – przy dobrze skonstruowanym procesie – zmusić zespół do pytania, dlaczego dana klasa błędów w ogóle powraca. Wtedy logbook, retrospektywa, baza wiedzy i mentoring tworzą razem organizacyjny metabolizm wiedzy.

Tutaj ujawnia się zasadnicze zadanie przywództwa ery AI: nie maksymalizować ilości inteligencji maszynowej, lecz zdolność organizacji do mądrego korzystania z inteligencji, której sama nie musi już w całości wytwarzać. To różnica między posiadaniem zasobu a posiadaniem instytucji zdolnej go wykorzystać. Państwo może mieć ropę i pozostać biedne; firma może mieć najlepszy model na świecie i pozostawać źle zarządzana. Sukces transformacji nie będzie należał do przedsiębiorstw z największą liczbą agentów, lecz do tych, które zachowają właściwe napięcie między automatyzacją a praktyką, szybkością a refleksją, pomocą a samodzielnością oraz delegowaniem a odpowiedzialnością.

AI może być nauczycielem, ale nie powinno odbierać uczniowi prawa do trudności. Może być współpracownikiem, ale nie powinno pozbawiać człowieka fachowości potrzebnej do sprzeciwu. Może być wykonawcą, ale organizacja nie może przestać rozumieć wykonywanej pracy. Z tej perspektywy pytanie „Kto wychowa seniora?" okazuje się pytaniem o całe przyszłe przedsiębiorstwo. Odpowiedź nie brzmi: AI. Nie brzmi też: człowiek bez AI.

## Ekonomia AI-first to przesunięcie kosztów z generowania na akceptację zmiany

Senior przyszłości będzie najprawdopodobniej produktem celowo zaprojektowanego środowiska hybrydowego. To przestrzeń, w której maszyna przyspiesza uczenie, ale nie eliminuje koniecznej praktyki; senior przekazuje kontekst, lecz nie monopolizuje wiedzy; junior korzysta z automatyzacji, choć jest okresowo zmuszany do samodzielnego rozwiązywania problemów; a lider chroni nie tylko bieżący output, lecz także zdolność zespołu do wydawania trafnych sądów za pięć czy dziesięć lat.

Dopiero na takim fundamencie można sensownie mówić o ekonomii adopcji. Jeśli bowiem przedsiębiorstwo policzy wyłącznie oszczędzone godziny, licencje, tokeny i wzrost velocity, pomijając koszt szkoleń, review, utrzymania kompetencji, długu technicznego, błędów oraz zależności od dostawcy, otrzyma bardzo dokładny wynik dla źle postawionego pytania.

Ekonomiczna opowieść o sztucznej inteligencji w programowaniu zwykle zaczyna się od kuszącego rachunku: jeśli deweloper z AI wykonuje zadanie szybciej, firma otrzymuje więcej rezultatu z tej samej godziny pracy. W efekcie produktywność rośnie, koszt jednostkowy maleje, a inwestycja niemal automatycznie się zwraca. Intuicja ta jest jednak prawdziwa tylko pod warunkiem, że poprawnie zdefiniujemy zarówno „rezultat", jak i „koszt".

W świecie agent-centric SDLC właśnie te dwie kategorie stają się problematyczne. Kod wygenerowany nie jest jeszcze kodem dostarczonym; kod dostarczony nie jest jeszcze kodem wartościowym, a ten ostatni nie musi pozostać ekonomicznie użyteczny przez cały okres swojego życia. Rachunek adopcji AI zaczyna się więc nie od ceny tokena, lecz od pytania, co właściwie przedsiębiorstwo kupuje za jego zużycie. Cassani słusznie próbuje wyjść poza koszt samych licencji.

Jego model budżetu AI-Augmented przewiduje nową kategorię wydatków na API i tokeny, zwiększenie nakładów na szkolenia i upskilling, wzrost znaczenia narzędzi własnych oraz rurociągów agentowych, a także większe koszty bezpieczeństwa, audytu i compliance. W tym modelu udziały przeznaczone na szkolenia przesuwają się z 5–10% do 15–20%, na rozwój narzędzi

własnych z 10–15% do 20–25%, a na mitygację ryzyka z około 5% do 10–15% budżetu. Liczb tych nie należy traktować jako empirycznej normy.

Znacznie ważniejsza jest ich struktura: adopcja AI nie tyle usuwa koszty IT, co przemieszcza je między kategoriami. To zasadnicza różnica wobec narracji redukcyjnej, w której narzędzie za kilkadziesiąt euro miesięcznie ma zastąpić część kosztu inżyniera liczonego w tysiącach. Licencja jest ceną wejścia, a nie całkowitym kosztem posiadania zdolności. Samochód również kosztuje więcej niż cena zakupu: wymaga paliwa, serwisu, ubezpieczenia i kierowcy.

W przedsiębiorstwie agentowym podobną funkcję pełnią integracja, observability, evals, guardrails, szkolenia, utrzymywanie bibliotek kontekstów oraz czas ludzi poświęcony na walidację efektów pracy maszyny. Dlatego ekonomika AI powinna przesunąć się z rachunku cost of generation ku rachunkowi total cost of accepted change. Najważniejsze pytanie nie brzmi: ile kosztuje wygenerowanie tysiąca linii kodu? Tysiąc linii może być aktywem albo zobowiązaniem. Istotne jest raczej, ile kosztuje doprowadzenie poprawnej, bezpiecznej i utrzymywalnej zmiany od intencji biznesowej aż do produkcji oraz jej późniejszy serwis. W takim rachunku token jest tylko jednym ziarnkiem piasku.

Koszt review seniora, regresji, opóźnionego wykrycia błędu czy przyszłej refaktoryzacji może wielokrotnie przewyższyć cenę wygenerowania rozwiązania. Materiał Cassaniego proponuje klasyczny rachunek ROI, w którym od całkowitej wartości wygenerowanej przez AI odejmuje się pełen koszt inwestycji. Autor zalicza po stronie korzyści produktywność i skrócenie time-to-market, a po stronie kosztów licencje, szkolenia i integrację. Sam sposób myślenia jest poprawny, jednak problem pojawia się w momencie wpisania do wzoru nadmiernie optymistycznych założeń, którym precyzyjna arytmetyka nadaje pozór rzeczywistości.

Źródło podaje przykład dziesięcioosobowego zespołu, dla którego inwestycja pierwszego roku wynosi 34 900 euro, a przypisana jej roczna wartość 476 000 euro (z czego większość wynika ze wzrostu produktywności). Przy takich założeniach Cassani otrzymuje ROI około 1263,9 procent i okres zwrotu rzędu 1,1 miesiąca. To jednak sytuacja, w której najtańszy kod może okazać się najdroższym kodem w całym cyklu jego życia.

## Pułapka ekstrapolacji ROI w adopcji AI

Matematycznie rachunek jest spójny, jednak metodologicznie należy powiedzieć wyraźnie: to scenariusz Cassandry oparty na założeniach dotyczących produktywności, jakości i wartości czasu, a nie uniwersalnie potwierdzona stopa zwrotu z wdrożenia AI. Rozróżnienie to jest kluczowe, gdyż

empiryczne wyniki badań nad wydajnością programistów wspieranych przez AI są skrajnie zależne od charakteru zadania. W kontrolowanym eksperymencie Microsoft Research i GitHub programiści korzystający z Copilota napisali serwer HTTP średnio o 55,8 procent szybciej niż grupa kontrolna. Było to badanie randomizowane, ale oparte na dobrze ograniczonym, stosunkowo krótkim zadaniu. Nie oznacza to, że wartość 55,8 procent można bezkrytycznie wpisać do każdego arkusza inwestycyjnego CTO.

Kilka lat później METR przeprowadził eksperyment o zupełnie innej konstrukcji. Doświadczeni maintainerzy wykonywali rzeczywiste zadania we własnych repozytoriach open source, które znali od lat. W tych warunkach narzędzia AI z początku 2025 roku nie przyspieszyły ich pracy; średnio czas realizacji wzrósł o 19 procent. Nie oznacza to jednak, że AI "spowalnia programistów". Wskazuje raczej na coś ciekawszego: wynik zależy od relacji między kompetencją człowieka, znajomością repozytorium, charakterem zadania, jakością narzędzia a kosztem jego kontrolowania. Jeszcze bardziej pouczająca jest aktualizacja METR z lutego 2026 roku.

Badacze odnotowali sygnały, że nowsze narzędzia mogły już przyspieszać część deweloperów, lecz sami autorzy uznali te oszacowania za słaby dowód ze względu na efekty selekcji i trudności pomiarowe – zwłaszcza gdy programiści zaczęli używać wielu agentów równocześnie. Nauka zachowuje się tu dokładnie tak, jak powinna: zamiast ogłaszać wygodną prawdę o "x procent wzrostu produktywności", pokazuje dynamicznie zmieniający się układ zależności. Stałą nie jest liczba, lecz konieczność mierzenia własnego procesu.

Z tego względu centralnym błędem ekonomicznym adopcji AI może stać się ROI by extrapolation: przedsiębiorstwo bierze rezultat uzyskany przez kogoś przy jednym zadaniu, mnoży go przez liczbę pracowników, dni i średni koszt wynagrodzenia, a następnie ogłasza milionowe oszczędności, zanim ktokolwiek sprawdzi, czy zaoszczędzona godzina rzeczywiście przełożyła się na dodatkową wartość. Czas zaoszczędzony nie jest automatycznie przychodem. Jeśli pracownik skończy zadanie trzydzieści minut wcześniej, lecz ten czas zostanie pochłonięty przez dodatkowy review, spotkanie, poprawki lub zwykłe zwiększenie liczby otwartych ticketów, firma nie zyskała wartości równej trzydziestu minutom jego stawki godzinowej. Ekonomista powinien w tym miejscu zapytać o krańcową wartość uwolnionego czasu.

Jeżeli skrócenie developmentu pozwala na wcześniejsze wejście produktu na rynek, uniknięcie kary kontraktowej lub obsłużenie większej liczby klientów – czas ma mierzalną wartość. Jeśli natomiast zwiększona prędkość produkowania kodu prowadzi jedynie do większego backlogu review, mamy do czynienia z lokalną poprawą produktywności bez wzrostu przepustowości całego systemu. To klasyczna teoria ograniczeń Goldratta: przyspieszenie stanowiska, które nie jest wąskim gardłem, nie zwiększy produkcji całej fabryki; może jedynie powiększyć zapas oczekujący przed kolejnym

etapem. Agentowa fabryka oprogramowania może właśnie w ten sposób produkować zapasy kodu. Generator pracuje szybciej, więc powstaje więcej pull requestów, większe diffy i większa liczba wariantów.

## ROI z AI wynika z architektury organizacji, a nie z modelu

Wąskim gardłem przesuwa się wtedy ku review, walidacji, testom bezpieczeństwa, decyzjom architektonicznym i integracji. Najcenniejszym zasobem organizacji przestaje być zdolność generowania rozwiązania; staje się nim ograniczona liczba minut seniora zdolnego powiedzieć: "to rozwiązanie wygląda poprawnie, ale jest błędne z powodów, których test nie widzi". DORA w raporcie z 2025 roku opisuje AI jako wzmacniacz istniejącego systemu organizacyjnego – narzędzia amplifikują zarówno cechy organizacji wysokiej jakości, jak i dysfunkcje organizacji słabej. Badanie to oparto na grupie pięciu tysięcy respondentów oraz ponad stu godzinach materiału jakościowego.

Dla ekonomii adopcji ma to kapitalne znaczenie. ROI nie jest właściwością modelu. ROI jest właściwością układu model-człowiek-proces-architektura-ryzyko. Ta sama technologia może generować wartość w jednym przedsiębiorstwie i koszt w drugim. Z tego powodu DORA opracowała AI Capabilities Model, w którym samo posiadanie narzędzi nie jest traktowane jako wystarczający warunek sukcesu; kluczowe są techniczne i kulturowe zdolności organizacji pozwalające wykorzystać AI bez degradacji jakości procesu. Jest to w istocie odpowiednik starej ekonomicznej prawdy dotyczącej technologii ogólnego zastosowania. Komputer osobisty nie zwiększał produktywności samym faktem postawienia go na biurku, a elektryfikacja nie polegała jedynie na zastąpieniu silnika parowego elektrycznym w niezmienionej fabryce. Dopiero przeprojektowanie organizacji pozwalało w pełni wykorzystać nowe możliwości.

Robert Solow zauważył kiedyś ironicznie, że erę komputerów widać wszędzie poza statystykami produktywności. Paradoks Solowa powinien być ostrzeżeniem również dla epoki AI. Technologia może rozpowszechniać się szybciej, niż organizacje potrafią przebudować procesy, kompetencje i modele zarządzania wokół niej. W pierwszej fazie inwestycji przedsiębiorstwo ponosi koszty uczenia się, integracji i reorganizacji, podczas gdy pełne korzyści przychodzą później – lub nie przychodzą wcale, jeśli firma automatyzuje stary proces zamiast zaprojektować nowy. Cassani intuicyjnie wychwytuje tę potrzebę, proponując budżet portfelowy: część środków trafia na sprawdzone zastosowania, część na eksperymenty, a część na szkolenia i narzędzia własne. Warto rozszerzyć tę koncepcję o język ekonomii opcji realnych. Eksperyment z AI nie musi być od razu projektem mającym wykazać dodatni ROI; może być opcją na przyszłą kompetencję. Firma ponosi

ograniczony koszt pilotażu, aby sprawdzić, czy określony wzorzec pracy ma sens, zanim zamrozi większy kapitał w architekturze agentowej. Takie podejście chroni przed dwoma symetrycznymi błędami.

Pierwszy polega na adopcji totalnej: "AI jest przyszłością, więc wdrażamy wszędzie". Drugi na konserwatyzmie absolutnym: "nie znamy dokładnego ROI, więc nie inwestujemy". Zarządzanie portfelowe pozwala rozmieszczać zadania na różnych poziomach ryzyka i dojrzałości. Stabilne, dobrze mierzalne zastosowania mogą być skalowane, niepewne pozostają eksperymentami, a krytyczne wymagają rygorystycznego dowodu bezpieczeństwa. Innymi słowy: przedsiębiorstwo nie obstawia całego kapitału na jedną prognozę technologicznego jutra. Nową kategorią ekonomiczną stają się tokeny. W klasycznym oprogramowaniu marginalny koszt wykonania raz napisanego algorytmu bywa znikomy. W systemach opartych na komercyjnych modelach każda interakcja generuje mierzalny koszt inferencji. W modelu Cassaniego wydatki na tokeny i API pojawiają się jako nowa, dynamiczna kategoria budżetowa, której w tradycyjnym IT praktycznie nie było.

## Ekonomika agentowa to zarządzanie wartością, nie tokenami

Sama cena miliona tokenów jest jednak miernikiem jeszcze bardziej zwodniczym niż koszt linii kodu. Model tańszy w przeliczeniu na token może wymagać więcej iteracji, generować więcej błędów i wymuszać droższy proces review. Z kolei model droższy może rozwiązać zadanie za pierwszym razem. Ekonomiczną jednostką miary nie powinien więc być token, lecz zaakceptowane zadanie o określonej jakości.

W systemach agentowych problem ten narasta, ponieważ agent rzadko wykonuje pojedyncze wywołanie. Planuje, czyta, korzysta z narzędzi, ponawia próby, testuje i koryguje rezultat. Systemy wieloagentowe dodatkowo mnożą liczbę takich trajektorii. Anthropic, opisując własny system wieloagentowego researchu, wskazuje, że w jego danych zwykły agent zużywał około czterokrotnie więcej tokenów niż standardowa interakcja czatowa, a system wieloagentowy – około piętnastokrotnie więcej.

Firma wprost zaznacza, że taka architektura ma ekonomiczny sens przede wszystkim w zadaniach na tyle wartościowych, by wzrost jakości uzasadniał koszt. Choć są to dane konkretnego systemu badawczego, a nie uniwersalny mnożnik dla wszystkich agentów, płynąca z nich lekcja jest szersza: autonomia zużywa budżet. Można zatem sformułować zasadę proporcjonalności wysiłku inferencyjnego do wartości zadania. Nie należy angażować rady ekspertów, pięciu agentów i

miliona tokenów po odpowiedź, którą deterministyczny skrypt zwróci w milisekundę. Agentowość staje się ekonomicznie uzasadniona tam, gdzie niepewność, złożoność lub potencjalna wartość rezultatu równoważą koszt poznawczy maszyny. Anthropic opisuje podobne podejście praktyczne: liczba subagentów i wywołań narzędzi powinna rosnąć wraz ze złożonością zapytania, gdyż jednym z wczesnych problemów systemu było nadmierne inwestowanie zasobów w proste zadania. To agentowy odpowiednik zasady, że nie wysyła się helikoptera po gazetę.

Koszt inferencji nie jest jednak parametrem stałym. Wraz z kolejnymi generacjami modeli zmieniają się zarówno ceny, jak i efektywność wykorzystania tokenów. Badania Anthropic nad agentami w domenie smart contracts wykazały znaczący spadek liczby tokenów potrzebnych do uzyskania skutecznego wyniku wraz z rozwojem modeli. Nie należy z tego wyprowadzać uniwersalnej stopy spadku kosztów dla programowania, lecz w rachunku strategicznym trzeba uwzględnić, że architektura zbyt mocno związana z dzisiejszym modelem cenowym może szybko stać się nieaktualna. Planowanie AI wymaga więc scenariuszy, a nie jednego trzyletniego arkusza udającego pewność.

Ekonomiczny sens ma także inwestowanie w evals, mimo że początkowo jawią się one jako czysty koszt. Anthropic zwraca uwagę, że dobrze zbudowany system ewaluacji pozwala mierzyć nie tylko trafność agentów, ale także koszt zadania, zużycie tokenów, latency i błędy; ponadto ułatwia szybką ocenę nowych modeli, eliminując konieczność każdorazowego przeprowadzania tygodni ręcznych testów. W języku finansowym eval suite jest więc kapitałem pomiarowym. Sam w sobie niczego nie sprzedaje, ale obniża przyszły koszt podejmowania decyzji technologicznych.

Analogicznie należy wyceniać guardrails, sandboxy, audit logi i rollback. W rachunku kosztowym wyglądają one jak narzut; w rachunku ryzyka są opcją ograniczającą straty ogonowe. Ubezpieczenie przeciwpożarowe również nie zwiększa codziennego obrotu restauracji, lecz byłoby absurdem uznać je za inwestycję bez wartości tylko dlatego, że większość dni kończy się bez pożaru. Podobnie koszt AI governance nie powinien być oceniany wyłącznie przez pryzmat codziennego wzrostu velocity. Jego ekonomicznym produktem jest zmniejszenie prawdopodobieństwa lub rozmiaru zdarzenia, którego przedsiębiorstwo chce uniknąć. Źródło Cassaniego słusznie włącza redukcję ryzyka do strony korzyści ROI oraz bezpieczeństwo i compliance do kosztów inwestycji. W praktyce najtrudniejsze pozostaje jednak przypisanie wartości zdarzeniu, które nie nastąpiło.

CFO widzi fakturę za audyt, ale nie widzi incydentu, któremu ten zapobiegł. Stąd naturalna presja organizacyjna, by optymalizować koszty bezpieczeństwa aż do dnia, w którym zmaterializuje się ryzyko. Dojrzały model ekonomiczny powinien zatem posługiwać się nie tylko średnią oczekiwaną wartością, ale także wartością ekspozycji na zdarzenia rzadkie i bardzo kosztowne. Podobnego rachunku wymaga dług techniczny.

# Ekonomia długu technicznego i koszt ludzkiej weryfikacji w erze AI

Materiał Cassaniego opisuje zjawisko "feature creep" – skłonność agentów do dodawania niezamówionych udogodnień oraz tendencję do mnożenia plików i struktur zamiast upraszczania legacy code. Autor proponuje metaforyczny system „kredytów długu technicznego", którego przekroczenie wymuszałoby okres refaktoryzacji. Można spierać się z arbitralnością takiego rozwiązania, lecz jego ekonomiczna intuicja jest trafna: łatwość generowania kodu kusi, by finansować dzisiejsze velocity przyszłym kosztem utrzymania. Ward Cunningham ukuł metaforę długu technicznego właśnie po to, by kod można było rozumieć w kategoriach ekonomicznych.

Szybsze rozwiązanie jest racjonalne wtedy, gdy organizacja świadomie akceptuje przyszłe „odsetki". Problem pojawia się, gdy dług zaciągany jest bez ewidencji. AI może drastycznie zwiększyć skalę tego zadłużenia, ponieważ niezwykle tanio produkuje rozwiązania działające lokalnie. Jeśli koszt stworzenia kolejnej abstrakcji spada niemal do zera, ekonomicznym hamulcem musi stać się koszt jej późniejszego utrzymania. Stąd paradoks: im tańsze staje się pisanie kodu, tym bardziej opłacalne może być jego niepisanie. Najlepsza odpowiedź agenta na ticket nie zawsze powinna brzmieć „oto implementacja". Czasem najwyższą wartość wnosi informacja, że funkcjonalność już istnieje, problem można rozwiązać konfiguracją, warto usunąć kod zamiast dodawać nowy lub specyfikacja opisuje potrzebę, która nie uzasadnia technicznej komplikacji. W świecie spadających kosztów produkcji zdolność do odmowy produkowania zyskuje premię ekonomiczną. Prowadzi to do idei return on deletion. Tradycyjna kultura inżynieryjna chętnie celebruje powstanie nowego systemu, lecz rzadko nagradza usunięcie pięciu zbędnych usług. AI może pogłębić tę asymetrię, jeśli KPI będą mierzyć liczbę wykonanych zadań lub wygenerowanych funkcji. Dojrzałe przedsiębiorstwo powinno zatem premiować redukcję złożoności, konsolidację oraz zmniejszenie powierzchni utrzymania.

Kod, którego nie ma, nie wymaga patchowania. Inną ukrytą kategorią kosztów jest human verification tax. Każdy rezultat AI wymagający ludzkiego sprawdzenia konsumuje uwagę. Jeśli maszyna generuje rozwiązanie w trzy minuty, a senior potrzebuje czterdziestu minut na jego rzetelny audyt, realny koszt ekonomiczny nie wynosi trzech minut.

W najlepszym razie AI zmieniła strukturę pracy seniora; w najgorszym – stworzyła dodatkowy obowiązek. Sytuacja staje się niebezpieczna, gdy zarząd wylicza oszczędności czasu programisty generującego kod, pomijając dodatkowe obciążenie osób dokonujących review. Konieczne jest zatem wprowadzenie miernika net cognitive savings: ile ludzkiej uwagi system rzeczywiście

oszczędza po odjęciu czasu potrzebnego na formułowanie poleceń, dostarczanie kontekstu, weryfikację, poprawki i analizę błędów.

## Wartość biznesowa zamiast mechanicznej produktywności

Strategia ta nie musi przyjmować jednej, uniwersalnej formuły. Wystarczy, że przedsiębiorstwo przestanie utożsamiać szybszą generację kodu z oszczędnością netto. Kolejnym kosztem jest vendor lock-in, znacznie subtelniejszy od klasycznego uzależnienia od platformy chmurowej. Agent może zostać głęboko osadzony w firmowych promptach, konfiguracjach, pamięci, narzędziach, formatach kontekstu, evalach i samym sposobie pracy zespołów. W takim układzie migracja przestaje oznaczać zwykłą zmianę endpointu API.

Może ona wymagać przebudowy zachowań agentów, ponownej walidacji bezpieczeństwa, dostosowania bibliotek kontekstowych oraz przeszkolenia pracowników. Ekonomia AI-first powinna zatem wyceniać switching cost inteligencji, a nie tylko koszt infrastruktury. Najlepszą ochroną przed tym ryzykiem nie musi być unikanie dostawców komercyjnych, lecz architektura zachowująca rozdział pomiędzy firmowym kontekstem, evalami i procesami a konkretnym modelem. DORA podkreśla znaczenie zdolności organizacyjnych, a nie samego narzędzia; z kolei rozwój evals umożliwia porównywanie nowych modeli pod kątem własnych zadań, zamiast wiązania przedsiębiorstwa z rankingami producentów. Model powinien być wymienialnym elementem systemu tak długo, jak ekonomicznie i technicznie jest to możliwe.

Należy wreszcie odróżnić zwykłą produktywność od produktywności wartościowej. Programista może dzięki AI zamknąć dwa razy więcej ticketów, ale jeżeli połowa z nich dotyczy funkcji, których klient nie potrzebuje, przedsiębiorstwo podwoiło prędkość marszu w złym kierunku. To stare prawo Druckera powraca z nową siłą: nie ma nic równie bezużytecznego jak bardzo efektywne wykonywanie rzeczy, których nie należało wykonywać. Agent nie rozwiązuje problemu priorytetyzacji biznesowej. Może wręcz zwiększyć jego znaczenie, ponieważ obniża techniczny koszt materializacji każdej zachcianki.

Time-to-market staje się prawdziwą wartością tylko wtedy, gdy wcześniejsze wejście na rynek ma realne znaczenie. Czasami tydzień przewagi generuje ogromną rentę. Innym razem przyspieszenie produktu, który wciąż wymaga miesięcy zmian regulacyjnych, sprzedażowych czy operacyjnych, nie ma wartości równej kosztowi tygodnia pracy zespołu. Rachunek Cassaniego słusznie uwzględnia time-to-market jako składnik korzyści, jednak każda organizacja musi przypisywać mu wartość na

podstawie własnego modelu biznesowego, a nie poprzez mechaniczne kopiowanie procentów ze studium przypadku.

Podobnie należy traktować przedstawiony przez Cassaniego scenariusz większego zespołu hybrydowego: 12 programistów współpracujących z 8-10 agentami, przy rocznym budżecie operacyjnym AI rzędu 280 tysięcy euro i zakładanym ROI od 180 do 220 procent. To scenariusz wdrożeniowy autora, użyteczny jako ilustracja sposobu budowania budżetu, a nie jako benchmark rynkowy. Prawdziwy test zaczyna się dopiero wtedy, gdy przedsiębiorstwo podstawia własne dane: rzeczywisty koszt review, własny defect rate, cycle time, wartość opóźnienia i koszt incydentu. Najbardziej dojrzałym KPI adopcji AI nie jest zatem pytanie o to, „ile kodu generuje AI" ani nawet „ile godzin oszczędzamy". Powinno ono brzmieć: ile dodatkowej wartości biznesowej uzyskujemy z jednostki całkowitego kosztu systemu przy zachowaniu akceptowalnego poziomu ryzyka, jakości i zdolności organizacyjnej. Dopiero taki rachunek pozwala zestawić ze sobą szybkość, pieniądze i bezpieczeństwo.

Równie ważna jest dynamika rozkładu tej wartości. AI może dramatycznie skrócić czas realizacji zadań łatwych, nie zmieniając niewiele w tych trudnych. Może zwiększyć produktywność juniora bardziej niż eksperta – lub odwrotnie, zależnie od domeny. Może tworzyć większą wartość przy greenfield development niż w skomplikowanym legacy, albo znakomicie sprawdzać się właśnie w określonych klasach modernizacji.

## Podejście problem-first jako fundament ekonomiki adopcji AI

Rozbieżność między eksperymentem GitHub Copilot a badaniem METR nie jest problemem, który należy „rozstrzygnąć". To raczej sygnał, że średnia globalna może być ekonomicznie mało użyteczna. Przedsiębiorstwo powinno zatem budować własną mapę krańcowej produktywności AI. Zamiast pytać, czy AI w ogóle „działa", warto zapytać: gdzie działa najlepiej, dla kogo, przy jakim rodzaju zadań, z jakim modelem nadzoru i przy jakiej wartości biznesowej? Jedno zastosowanie może uzasadniać użycie autonomicznego agenta. Drugie wystarczy obsłużyć przez AI-Assisted Coding. Trzecie może okazać się tańsze bez wsparcia sztucznej inteligencji.

Czwartemu lepiej posłuży zwykły, deterministyczny skrypt. Najdroższą innowacją bywa technologia użyta do problemu, którego prostsze narzędzie rozwiązywało znakomicie. Właśnie dlatego pojęcie AI-first wymaga drobnej, lecz istotnej rewizji. Dojrzała organizacja nie powinna być AI-first w sensie „najpierw spróbuj użyć AI". Powinna być problem-first, evidence-first i value-first. AI jest

jedną z dostępnych technologii – i może być niezwykle potężną – lecz celem przedsiębiorstwa nie jest maksymalizacja jej wykorzystania. Celem pozostaje tworzenie wartości przy rozsądnym zarządzaniu kapitałem, ludzką uwagą i ryzykiem.

W tej perspektywie najbardziej ekonomicznym agentem nie jest ten najtańszy czy najbardziej autonomiczny. Jest nim ten, który wystarczająco dobrze wykonuje właściwe zadanie przy najniższym całkowitym koszcie uzyskania akceptowalnego rezultatu. Czasami będzie to największy model, czasami mniejszy, a czasem zespół kilku agentów lub człowiek z prostym autocomplete. A zdarzą się sytuacje – co w epoce technologicznego entuzjazmu może być najtrudniejsze do zaakceptowania – gdy najlepszy ROI osiąga się, nie uruchamiając żadnego modelu. Tak rozumiana ekonomia adopcji AI obala kolejną iluzję. Kod przestaje być kosztowny dlatego, że trudno go napisać. Może stać się kosztowny dlatego, że zbyt łatwo go wyprodukować.

Kiedy podaż kodu eksploduje, wartość przesuwa się w stronę selekcji, upraszczania, walidacji i architektury. Maksyma Antoine'a de Saint-Exupéry'ego, według której doskonałość osiąga się nie wtedy, gdy nie można już nic dodać, lecz gdy nie można już nic odjąć, zaczyna brzmieć jak ekonomiczna zasada programowania agentowego.

Rachunku wartości nie można jednak domknąć bez zbadania technicznego długu AI: jakości generowanego kodu, feature creep, code churn, entropii repozytorium i nowej ekonomii utrzymania. Agent może w kilka godzin stworzyć system, którego budowa wcześniej trwała tygodnie. Prawdziwe pytanie pojawia się następnego dnia: kto będzie go rozumiał, poprawiał i utrzymywał – oraz ile firma zapłaci za każdą minutę zaoszczędzoną podczas jego narodzin? Ten wątek Fundacja Dobre Państwo rozwija w kolejnym artykule – zapraszamy.

> W świecie, w którym kod stał się tani i powszechny, największym luksusem i najcenniejszą kompetencją może okazać się umiejętność świadomej rezygnacji z jego pisania. Prawdziwa doskonałość inżynierii agentowej nie objawi się w liczbie funkcji dostarczonych przez maszyny, lecz w odwadze liderów, by nie dopuścić do produkcji rozwiązań, których nikt już nie potrafi wyjaśnić. Czy jesteśmy gotowi na świat, w którym najwyższą wartością programisty nie jest to, co potrafi stworzyć, lecz to, czego potrafi zakazać maszynie?

## Inspiracje

**[1]** **"Code Revealed"** Alexio Cassani

*2026 | Packt Publishing | ISBN: 978-1807789305*

**Alexio Cassani** to włoski przedsiębiorca, lider technologiczny i badacz z ponad dwudziestoletnim doświadczeniem w inżynierii oprogramowania i transformacji cyfrowej. Absolwent inżynierii komputerowej na Politechnice Mediolańskiej (2003), piastował ważne stanowiska kierownicze, w tym stanowisko dyrektora ds. technologii w firmach Cortilia i Webscience. Jest współzałożycielem FairMind, platformy koncentrującej się na zwiększaniu produktywności rozwoju oprogramowania za pomocą agentów AI. Prace Cassaniego łączą badania naukowe z praktycznymi zastosowaniami w przedsiębiorstwach, koncentrując się na zarządzaniu sztuczną inteligencją, przepływach pracy agentów oraz współpracy człowiek-maszyna. Jest uznanym liderem w dziedzinie generatywnej sztucznej inteligencji i cyfrowej transformacji przedsiębiorstw, przyczyniając się zarówno do praktyk branżowych, jak i projektów badawczych, takich jak te dotyczące asystentów konwersacyjnych opartych na sztucznej inteligencji w sektorze turystycznym.

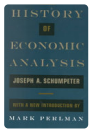
***Alexio Cassani*** *is an Italian entrepreneur, technology leader, and researcher with over twenty years of experience in software engineering and digital transformation. A graduate in Computer Engineering from Politecnico di Milano (2003), he has held significant leadership roles, including CTO at Cortilia and Webscience. He is the co-founder of FairMind, a platform focused on enhancing software development productivity through AI agents. Cassani's work bridges the gap between academic research and practical enterprise application, with a focus on AI governance, agentic workflows, and human-machine collaboration. He is a recognized thought leader in the field of generative AI and corporate digital transformation, contributing to both industry practices and academic research projects, such as those involving AI-powered conversational assistants in the tourism sector.*

*FairMind*

## Literatura uzupełniająca

### Książki

## Literatura pokrewna (Google Scholar):



**[1] "History of Economic Analysis"**

Joseph Schumpeter

Oxford University Press, USA | ISBN: 9780195105599

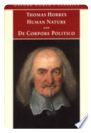
**[2] "History of Economic Analysis (Oxford Press, 1955)"**

Joseph Schumpeter

Taylor & Francis | ISBN: 9781134838714

**[3] "Theory of economic development"**

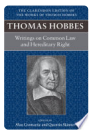
Joseph Schumpeter



**[4] "The Elements of Law"**

Thomas Hobbes

Oxford University Press, USA | ISBN: 9780192836823



**[5] "Thomas Hobbes: Writings on Common Law and Hereditary Right"**

Thomas Hobbes

2005 | Oxford University Press, USA | ISBN: 9780198237020

**[6] "Persian Letters"**

Charles Montesquieu

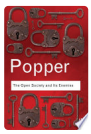
Palala Press | ISBN: 9781348042860

**[7] "Essay on Taste"**

Charles Montesquieu

**[8] "Arsace et Isménie"**

Charles Montesquieu




**[9] "The Open Society and Its Enemies"**

Karl Popper

Routledge | ISBN: 9781136700323

**[10] "In Search of a Better World"**

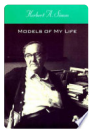
Karl Popper

1992 | Other

**[11] "Objective Knowledge"**

Karl Popper

Taylor & Francis | ISBN: 9781040621004



**[12] "Models of my life"**

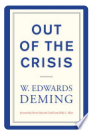
Herbert Simon

MIT Press | ISBN: 9780262326582

**[13] "Human Problem Solving"**

Herbert Simon

Prentice Hall




**[14] "Out of the Crisis"**

W. Edwards Deming

MIT Press | ISBN: 9780262535946

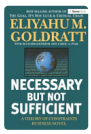
**[15] "The Choice"**

Eliyahu Goldratt

**[16] "The Goal"**

Eliyahu Goldratt

2004 | ISBN: 9780884271789




**[17] "Necessary But Not Sufficient"**

Eliyahu Goldratt

Gower Publishing Company, Limited | ISBN: 9780566084508

**[18] "Solow–Swan model"**

Robert Solow

**[19] "The Wiki Way"**

Ward Cunningham

Addison-Wesley Professional

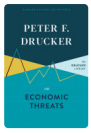
**[20] "The Wiki Way: quick collaboration on the Web"**

Ward Cunningham

Addison-Wesley Professional

[21] **"Agile Manifesto"**

Ward Cunningham



[22] **"Peter F. Drucker on Economic Threats"**

Peter F. Drucker

2020 | Harvard Business Press | ISBN: 9781633699601

[23] **"Managing in Turbulent Times"**

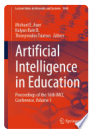
Peter Drucker

2012 | Routledge | ISBN: 9781136009143

[24] **"Management: Tasks, Responsibilities, Practices"**

Peter Ferdinand Drucker

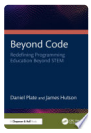
1974 | HarperCollins Publishers



[25] **"Artificial Intelligence in Education"**

Michael E. Auer

Springer Nature | ISBN: 9783032253873



[26] **"Beyond Code"**

Daniel Plate, James Hutson

2025 | CRC Press | ISBN: 9781040421666

[27] **"The Azure Cloud Native Architecture Mapbook"**

Stéphane Eyskens

2025 | Packt Publishing Ltd | ISBN: 9781805805045

[28] **"Fractal Large Language Model (FLLM)"**

SEYED RASOUL HAMZAH

2026 | SEYED RASOUL HAMZAH

[29] **"The Agentic Developer"**

Frank Westfield

2026 | Independently Published | ISBN: 9798195018009

[30] **"Prompt-Driven Development Handbook"**

Prasanna Venkatesan Nagarajan

2026 | BPB Publications | ISBN: 9789365892932

## Artykuły naukowe

### Autorzy przywoływani:


**[1] "Leviathan: With selected variants from the Latin edition of 1668"**

Thomas Hobbes, Edwin Curley

2021

Google Scholar

**[2] "The Elements of Law, Natural and Politic"**

William Reynolds Vance, Thomas Hobbes, Ferdinand Tönnies

1929 | The Yale Law Journal | DOI: 10.2307/790526

Google Scholar

**[3] "A Dialogue between a Philosopher and a Student, of the Common Laws of England"**

Thomas Hobbes

1673 | Oxford University Press eBooks | DOI: 10.1093/oseo/instance.00006683

Google Scholar

**[4] "Management of Software Projects"**

Frederick Brooks

2003 | Software Engineering for Image Processing Systems | DOI: 10.1201/9780203496107.ch9

Google Scholar

**[5] "Observations and Comments on the Organization Studies of the Systems Research Laboratory"**

Herbert Simon

1952 | RAND Corporation eBooks | DOI: 10.7249/rm922

Google Scholar

**[6] "The Cost of Paying Attention: Cognitive Resource Scarcity and Investor Activity Around FDA Announcements"**

Herbert Simon, Doron Kliger, Ofir Levi, Tiran Rothman

2014

Google Scholar

**[7] "Computational intelligence in economics and finance: Carrying on the legacy of"**

Herbert Simon, Shu-Heng Chen

2008

Google Scholar

[8] **"Chapter 3. From Information to Knowledge What information consumes is rather obvious: it consumes the attention of its recipients. Hence, a wealth of information creates a poverty of attention, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it."**

Herbert Simon

1999

Google Scholar

[9] **"Calidad, productividad y competitividad: la salida de la crisis"**

W. Edwards Deming, Jesús Nicolau Medina, Mercedes Gozalbes Ballester

1989 | Dialnet (Universidad de la Rioja)

Google Scholar

[10] **"A Further Account of the Idiots Savants, Experts with the Calendar"**

WILLIAM A. HORWITZ, W. Edwards Deming, Robert F. Winter

1969 | American Journal of Psychiatry | DOI: 10.1176/ajp.126.3.412

Google Scholar

[11] **"A System of Profound Knowledge"**

W. Edwards Deming

1998 | Elsevier eBooks | DOI: 10.1016/b978-0-7506-7009-8.50015-x

Google Scholar

[12] **"1 Transformation of western style of management"**

W. Edwards Deming

1988 | Handbook of statistics | DOI: 10.1016/s0169-7161(88)07003-8

Google Scholar

[13] **"Knowledge for Action: A Guide to Overcoming Barriers to Organizational Change"**

Chris Argyris

1993 | Virtual Defense Library (Ministerio de Defensa)

Google Scholar

[14] **"Single-Loop and Double-Loop Models in Research on Decision Making"**

Chris Argyris

1976 | Administrative Science Quarterly | DOI: 10.2307/2391848

Google Scholar

[15] **"Overcoming Organizational Defenses: Facilitating Organizational Learning"**

Chris Argyris

1990 | Medical Entomology and Zoology

Google Scholar

[16] **"Organizational Learning II"**

Chris Argyris

1996 | Medical Entomology and Zoology

Google Scholar

[17] **"A Contribution to the Theory of Economic Growth"**

Robert M. Solow

1956 | The Quarterly Journal of Economics | DOI: 10.2307/1884513

Google Scholar

[18] **"Technical Change and the Aggregate Production Function"**

Robert M. Solow

1957 | The Review of Economics and Statistics | DOI: 10.2307/1926047

Google Scholar

[19] **"Models of Man--Social and Rational"**

Robert M. Solow, Herbert A. Simon

1958 | The Review of Economics and Statistics | DOI: 10.2307/1926487

Google Scholar

[20] **"Capital-Labor Substitution and Economic Efficiency"**

K. J. Arrow, Hollis B. Chenery, B. S. Minhas, Robert M. Solow

1961 | The Review of Economics and Statistics | DOI: 10.2307/1927286

Google Scholar

[21] **"Manifesto for agile software development"**

K. Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham

2001

Google Scholar

[22] **"The Wiki Way: Quick Collaboration on the Web"**

Bo Leuf, Ward Cunningham

2001

Google Scholar

[23] **"The WyCash portfolio management system"**

Ward Cunningham

1992 | DOI: 10.1145/157709.157715

Google Scholar

[24] **"Agile Software Development: The Cooperative Game"**

Alistair Cockburn, Ward Cunningham, Arie van Bennekum, Martin Fowler, D. A. Thomas

2006

Google Scholar

## Artykuły pokrewne (Google Scholar):

[25] **"zIA: a GenAI-powered local auntie assists tourists in Italy"**

Alexio Cassani, Michele Ruberl, Antonio Salis, Giacomo Giannese, Gianluca Boanelli

2024 | arXiv (Cornell University) | DOI: 10.48550/arxiv.2407.11830

Google Scholar

Treść tworzy Fundacja Dobre Państwo.
Redaguje i publikuje APA ONE, autorski system redakcyjny Fundacji oparty na AI.
APA ONE Publishing System
Fundacja Dobre Państwo © 2026
Article ID: 73d4d35d-df3c-4ac8-ac5e-5a30b9b16f9e