Wszyscy analizowali funkcje Claude Code. Nikt nie analizował jego architektury.

By K M
kwi 11, 2026

Wszyscy analizowali funkcje Claude Code. Nikt nie analizował jego architektury.

Pięćset tysięcy linii wyciekłego kodu źródłowego ujawnia, że fosą w narzędziach do kodowania AI nie jest model, lecz uprząż.

Han HELOIR YAN, Ph.D.

31 marca 2026 roku tysiące programistów na całym świecie zrobiło to samo: wprowadzili kod źródłowy Claude Code z powrotem do Claude i poprosili go o wyjaśnienie.

Flagowe narzędzie CLI firmy Anthropic właśnie wyciekło z całej swojej bazy kodu TypeScript, liczącej 512 000 wierszy, poprzez plik mapy źródłowej przypadkowo dołączony do pakietu npm. W ciągu kilku godzin internet skatalogował 44 flagi funkcji, system zwierzaków Tamagotchi z 18 gatunkami i mechaniką gacha oraz wewnętrzne nazwy kodowe, takie jak „Tengu", „Fennec" i „Penguin Mode".

Ale lista funkcji to nie wszystko. Wszyscy już napisali ten artykuł. Prawdziwa wartość tego przecieku to nie to, co Claude Code potrafi, ale sposób myślenia Claude Code. A fakt, że deweloperzy zapłacili Anthropic za token, aby zrozumieć jego własny produkt? To nie ironia. To jest teza.

Zanim zaczniemy!

Jeśli to pomoże Ci dostarczać lepsze systemy AI:

Klaśnij 50 razy (tak, potrafisz!) — algorytm Medium faworyzuje tę opcję, zwiększając widoczność dla innych osób, które następnie odkryją artykuł.

Obserwuj mnie na Medium , LinkedIn i zasubskrybuj , aby otrzymywać mój najnowszy artykuł

Uprząż jest produktem

Większość osób zakładała, że Claude Code to cienka powłoka CLI dla API Claude. Interfejs terminala, który wysyła monit, przesyła strumieniowo odpowiedź i obsługuje kilka operacji na plikach.

Wyciekłe źródło przedstawia inną historię. Anthropic faktycznie dostarczył 512 000 linii kodu TypeScript: niestandardowy renderer terminala React z podwójnie buforowanym wyjściem ekranowym i układem Yoga Flexbox, ponad 60 narzędzi z bramką uprawnień i opóźnionym wykrywaniem, system orkiestracji wielu agentów, który generuje i koordynuje równoległe agenty robocze, działający w tle mechanizm konsolidacji pamięci, który działa podczas snu użytkownika, oraz samonaprawiającą się pętlę zapytań, która nie ulega awariom.

To nie jest opakowanie. To system operacyjny dla agenta AI.

Społeczność Hacker News podzieliła się na dwa obozy. Jeden z nich zbagatelizował wyciek, posługując się metaforą kasyna: „Kod źródłowy automatu do gry nie ma znaczenia dla menedżera kasyna". Modelem są pieniądze. Interfejs wiersza poleceń jest jednorazowy. Drugi obóz uważał inaczej: modelem jest krupier. Uprząż to kasyno. A kasyna są bardzo trudne do zbudowania.

Liczby przemawiają za drugą opcją. Opus 4.6 jest dostępny dla każdego poprzez API w cenie 5/25 dolarów za milion tokenów. Jednak VentureBeat donosi, że sam Claude Code generuje 2,5 miliarda dolarów rocznych przychodów cyklicznych, z czego 80% pochodzi z wdrożenia w przedsiębiorstwach. Deweloperzy nie płacą za ten model.Płacą za elementy, które czynią model użytecznym: orkiestrację narzędzi, system uprawnień, zarządzanie kontekstem, odzyskiwanie danych po błędach.Jeśli to odrzucisz, otrzymasz kosztowne narzędzie do automatycznego uzupełniania.

Samoodniesieniowa pętla przychodów z 31 marca utwierdza nas w tym przekonaniu. Jeden z deweloperów zbudował serwer MCP specjalnie po to, aby ludzie mogli korzystać z Claude Code do interaktywnej eksploracji wyciekłego kodu. Inny korzystał z Claude Code o 4 rano, aby przenieść podstawową architekturę Claude Code do Pythona przed wschodem słońca. Zespół w Chinach stworzył 12-sesyjny program nauczania inżynierii wstecznej na podstawie wyciekłego kodu źródłowego. Deweloper Jingle Bell uchwycił ten moment jednym zdaniem: „Dzisiejsze przychody Claude'a pochodzą od wszystkich, którzy korzystają z Claude'a do analizy jego kodu źródłowego".

Uprząż generuje popyt na model. Model generuje dochód z uprzęży. To koło zamachowe jest produktem.

Poniżej znajdują się trzy wzorce architektoniczne wyodrębnione z wyciekłego źródła. Nie funkcje. Nie nazwy kodowe. Prymitywy inżynieryjne, które napędzają koło zamachowe.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Stos uprzęży jako produktu.

Wzorzec 1: Pętla zapytań samonaprawiających się

Najdroższą inżynierią w Claude Code nie jest sztuczna inteligencja, lecz obsługa błędów.

W sercu Claude Code leży pętla zapytań. Nie jest to prosty cykl żądanie-odpowiedź. Jest to `while(true)` maszyna stanów, która zarządza zmiennym obiektem stanu w iteracjach, z jednym nadrzędnym celem projektowym: nigdy nie wyświetlać użytkownikowi surowego błędu.

Każda iteracja przebiega według tej samej sekwencji. Najpierw wstępnie pobiera pamięć i umiejętności równolegle (a nie sekwencyjnie, co podwoiłoby opóźnienie). Następnie stosuje kompresję komunikatów, jeśli kontekst staje się zbyt duży. Następnie wywołuje API z obsługą strumieniowania. Następnie uruchamia wszystkie narzędzia żądane przez model. Na koniec sprawdza, czy kontynuować pętlę, czy zakończyć działanie. W każdym kroku pętla śledzi przyczynę braku zakończenia, przechowując przyczynę przejścia w stanie, aby kolejna iteracja mogła się dostosować.

Prawdziwa inżynieria tkwi w kaskadzie odzyskiwania po błędzie. Gdy coś zawiedzie, pętla nie ulega awarii. Przechodzi ona przez sekwencję coraz bardziej agresywnych strategii odzyskiwania:

1. **Mikrokompaktowanie.** Wytnij wiadomości o niskiej wartości z kontekstu, aby zwolnić tokeny.
2. **Zwijanie kontekstu.** Jeśli zagęszczenie nie wystarczy, zwijaj całe fragmenty konwersacji w podsumowania.
3. **Eskalacja tokena.** Jeśli budżet wyjściowy modelu zostanie wyczerpany w trakcie zadania, wstrzyknij niewidzialną metawiadomość („Wznów bezpośrednio, bez przeprosin, bez podsumowania”) i kontynuuj. Maksymalnie trzy kolejne próby przed ujawnieniem przyczyny zatrzymania.
4. **Powrót do modelu zapasowego.** Jeśli model podstawowy jest niedostępny, należy powrócić do modelu alternatywnego.
5. **Błąd powierzchniowy.** Użytkownik widzi błąd dopiero po wyczerpaniu wszystkich ścieżek odzyskiwania.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Maszyna stanów pętli zapytań

Jeden z programistów, który przeprowadził inżynierię wsteczną 12 wersji kodu Claude'a ze zminimalizowanego kodu JavaScript, odkrył, że 5,4% wszystkich wywołań narzędzi zostało porzuconych: model zażądał narzędzia, narzędzie zostało uruchomione, ale wynik nigdy nie został zwrócony. Pętla samonaprawiająca została zaprojektowana tak, aby absorbować dokładnie tego typu awarie bez wiedzy użytkownika.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Współbieżne przetwarzanie wsadowe narzędzi.

Wykonywanie narzędzi dodaje kolejną warstwę. Pętla nie uruchamia narzędzi pojedynczo. Dzieli wywołania narzędzi na partie na podstawie klasyfikacji bezpieczeństwa współbieżności. Każde narzędzie deklaruje, czy jego równoległe uruchomienie jest bezpieczne, za pomocą `isConcurrencySafe()`metody. Narzędzia tylko do odczytu (polecenia grep, glob, odczyt plików) działają współbieżnie, do 10 naraz. Narzędzia do zapisu (edycja plików, polecenia bash z efektami ubocznymi) działają szeregowo. Partie działają naprzemiennie: odczyt partii, zapis partii, odczyt partii. Wykonawca narzędzia strumieniowego może rozpocząć wykonywanie narzędzi, gdy model nadal generuje dane wyjściowe, nakładając na siebie obliczenia i operacje wejścia /wyjścia, aby zmniejszyć opóźnienie.

Sam system narzędzi został zaprojektowany w oparciu o tę samą zasadę oszczędzania zasobów. Spośród ponad 60 narzędzi Claude Code, tylko około 40 ładuje się przy każdym żądaniu. Pozostałe 18 jest oznaczone jako odroczone. Są one niewidoczne dla modelu, dopóki nie zostaną wyszukane za pomocą dedykowanego

narzędzia ToolSearchTool. Gdy model potrzebuje danej funkcjonalności (integracja LSP, tworzenie zadań w tle, harmonogramowanie cron), wyszukuje, otrzymuje schemat i wywołuje narzędzie w tej samej turze. Użytkownik nie widzi niczego niezwykłego. Okno kontekstowe pozostało jednak o 200 tys. tokenów mniejsze, ponieważ schematy narzędzi, których model nie potrzebował, nigdy nie trafiały do pamięci roboczej.

Nawet kolejność na liście narzędzi ma znaczenie. Narzędzia są sortowane alfabetycznie przed wysłaniem do API. To nie jest kwestia estetyki, a optymalizacji pamięci podręcznej. Kolejność alfabetyczna zapewnia identyczność listy narzędzi we wszystkich żądaniach, co maksymalizuje szybkość trafień w pamięci podręcznej. Brak trafienia w pamięć podręczną na liście narzędzi oznacza konieczność ponownego przetworzenia tysięcy tokenów definicji schematu. Sortowanie według nazwy zmienia to w stabilny prefiks.

Kontrintuicyjna lekcja dla każdego, kto tworzy systemy agentowe: niezawodność to nie funkcja, którą dodaje się po uruchomieniu pętli głównej. Pętla główna to system niezawodności. Silnik zapytań Claude Code ma 46 000 wierszy nie dlatego, że ścieżka bezpieczeństwa jest złożona, ale dlatego, że ścieżki odzyskiwania są.

Wzorzec 2: Obliczanie czasu snu

Najważniejszą rzeczą, jaką robi Claude Code jest to, co robi, gdy go nie używasz.

Każde narzędzie do kodowania AI ma ten sam problem. Pracujesz z nim godzinami, budując kontekst: decyzje architektoniczne, wzorce debugowania, polecenia kompilacji, osobiste preferencje. Potem zamykasz terminal. Kolejna sesja zaczyna się od zera. Model nie pamięta dnia wczorajszego. Powtarzasz się. Popełnia te same błędy. Kontekst, który zbudowałeś, znika.

Odpowiedzią Claude'a Code'a jest system o nazwie autoDream. Nazwa jest celowa. To Claude, śniący.

Pomiędzy sesjami Claude Code tworzy rozwidlonego podagenta, którego jedynym zadaniem jest konsolidacja pamięci. Podagent odczytuje katalog pamięci projektu, przegląda ostatnie logi sesji, identyfikuje nowe informacje warte zachowania i przepisuje pliki pamięci, aby były czystsze, dokładniejsze i bardziej przydatne na potrzeby następnej sesji. Monit systemowy dla tego podagenta mówi dokładnie, o co chodzi: „Wykonujesz sen, refleksyjne przeglądanie swoich plików pamięci. Syntetyzujesz to, czego się ostatnio nauczyłeś, w trwałe, dobrze zorganizowane wspomnienia, aby przyszłe sesje mogły szybko się zorientować”.

Sen nie działa, kiedy mu się podoba. Ma trzy bramki, które muszą zostać uruchomione, zanim rozpocznie się konsolidacja.

System wyzwalania trójbramkowego

Brama 1: Czas. Co najmniej 24 godziny od ostatniego snu. Zapobiega to nadmiernej konsolidacji podczas aktywnej pracy w ramach wielu krótkich sesji.

Bramka 2: Sesje. Co najmniej 5 sesji od ostatniego snu. To gwarantuje, że zgromadziło się wystarczająco dużo nowego sygnału, aby konsolidacja była opłacalna.

Brama 3: Blokada. Podagent musi uzyskać blokadę konsolidacyjną. Zapobiega to równoczesnym snom, jeśli aktywnych jest wiele instancji Kodu Claude'a.

Gdy wszystkie trzy bramy zostaną przekroczone, sen przechodzi przez cztery fazy.

Faza 1, Orientacja. Wypisz katalog pamięci. Przeczytaj plik indeksu (MEMORY.md). Przejrzyj istniejące pliki tematyczne, aby zrozumieć aktualny stan.

Faza 2: Zbieranie sygnałów. Przeszukaj najnowsze źródła w poszukiwaniu nowych informacji wartych utrwalenia. Kolejność priorytetów: najpierw dzienniki, następnie wspomnienia (zmienione fakty), a na końcu przeszukaj transkrypty w poszukiwaniu wzorców, które model zauważył podczas pracy.

Faza 3: Konsolidacja. Zapis lub aktualizacja plików pamięci. Konwersja dat względnych („wczoraj”) na daty bezwzględne („30 marca 2026 r.”). Usuwanie faktów sprzecznych z nowszymi informacjami. Scalanie zbędnych wpisów.

Faza 4, Oczyszczanie i indeksowanie. Utrzymuj plik MEMORY.md poniżej 200 wierszy i około 25 KB. Usuń nieaktualne wskaźniki. Rozwiąż sprzeczności. Indeks musi pozostać na tyle mały, aby można go było załadować do kontekstu każdej przyszłej sesji bez znaczącego kosztu tokena.

Subagent Dream ma dostęp tylko do odczytu w bash. Może obserwować Twój projekt (odczyt plików, wyświetlanie katalogów, sprawdzanie stanu Gita), ale nie może niczego modyfikować. To granica bezpieczeństwa: proces konsolidacji nigdy nie powinien mieć skutków ubocznych na Twoją bazę kodu.

Ta architektura nawiązuje do koncepcji z badań Uniwersytetu Kalifornijskiego w Berkeley nad obliczeniami w czasie snu: wykorzystania bezczynnych cykli obliczeniowych do poprawy wydajności wnioskowania w przyszłości. W artykule zaproponowano wstępne wnioskowanie o prawdopodobnych przyszłych zapytaniach na podstawie zgromadzonego kontekstu. AutoDream patrzy wstecz, a nie w przyszłość. Organizuje on przeszłe wspomnienia, a nie przyszłe przewidywania. Filozofia projektowania jest jednak taka sama: inwestuj w obliczenia, gdy użytkownik nie czeka, aby następna sesja rozpoczęła się szybciej i z lepszym kontekstem.

Rezultatem jest czterowarstwowa architektura pamięci, jakiej nie oferowało żadne inne narzędzie do kodowania AI.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Czterowarstwowa architektura pamięci

Warstwa 1: CLAUDE.md. Instrukcje, które piszesz. Statyczne, tworzone przez człowieka, zawsze ładowane.

Warstwa 2: Pamięć automatyczna. Notatki, które Claude sporządza podczas każdej sesji. Polecenia kompilacji, spostrzeżenia dotyczące debugowania, decyzje dotyczące architektury, Twoje preferencje. Gromadzenie danych odbywa się automatycznie.

Warstwa 3: Pamięć sesji. Ciągłość konwersacji w ramach jednej sesji. Standardowe zarządzanie oknami kontekstowymi.

Warstwa 4: Auto Dream. Okresowa konsolidacja wszystkiego, co zgromadziło się w warstwach od 1 do 3. Zbieracz śmieci. Defragmentator. Faza snu REM.

Instrukcja obsługi, funkcja robienia notatek, funkcja pamięci krótkotrwałej i faza REM snu. To nie jest lista funkcji. To architektura poznawcza.

Wzorzec 3: Eliminacja cech w czasie kompilacji

Mapa źródłowa wyciekła właśnie dlatego, że system bezpieczeństwa działał. Kod został usunięty z kompilacji. Po prostu nie został usunięty z mapy.

Anthropic dostarcza jedną bazę kodu dwóm grupom odbiorców. Pracownicy wewnętrzni otrzymują KAIROS (zawsze aktywny asystent), BUDDY (towarzysz Tamagotchi), tryb koordynatora (orkiestracja wieloagentowa), tryb głosowy, tryb mostka i kilkanaście innych eksperymentalnych podsystemów. Użytkownicy zewnętrzni nie otrzymują żadnego z nich. Kompilacja zewnętrzna nie zawiera martwego kodu za instrukcjami warunkowymi. Kod jest fizycznie nieobecny w pliku wykonywalnym.

Mechanizmem jest `feature()`funkcja Buna, która jest obliczana w czasie kompilacji, a nie w czasie wykonywania. Gdy Anthropic buduje pakiet zewnętrzny, każde `feature('KAIROS')`wywołanie jest interpretowane `false`jako stała w czasie kompilacji. Bundler Buna eliminuje wtedy całą gałąź, usuwając martwy kod. Wynikowy plik JavaScript nie zawiera śladu KAIROS, BUDDY ani żadnego innego podsystemu z bramkami. Żadnych odwołań do ciągów znaków. Żadnych ciał funkcji. Żadnych ścieżek importu. Zniknęło.

Jest to pierwszy poziom dwupoziomowego systemu funkcji.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Bramkowanie funkcji w czasie kompilacji i w czasie wykonania.

Drugim poziomem jest bramkowanie w czasie wykonywania (runtime gateing) za pośrednictwem platformy flag funkcji GrowthBook. Flagi w czasie wykonywania (prefiksowane `tengu_`w bazie kodu) kontrolują stopniowe wdrażanie, testy A/B i wyłączniki awaryjne (kill switch) dla funkcji, które przeszły już bramki w czasie kompilacji. Funkcja sprawdzająca te flagi nosi nazwę `getFeatureValue_CACHED_MAY_BE_STALE()`. Ta nazwa jest decyzją projektową zakodowaną w kodzie: nieaktualne dane są akceptowalne dla bramek funkcji. Szybkość jest ważniejsza niż aktualność. Agent nigdy nie powinien blokować się podczas sprawdzania flag.

Te dwa poziomy służą różnym celom. Eliminacja w czasie kompilacji stanowi granicę bezpieczeństwa: funkcje dostępne wyłącznie wewnętrznie nie mogą w ogóle występować w artefakcie zewnętrznym, nawet jako niedostępny kod. Bramkowanie w czasie wykonywania to mechanizm wdrażania: funkcje, które są dopuszczone do wydania zewnętrznego, mogą być włączane stopniowo, monitorowane i natychmiast wyłączane w przypadku awarii.

Trzecia warstwa znajduje się na wierzchu obu: `USER_TYPE === 'ant'`funkcje bramek dostępne wyłącznie dla pracowników Anthropic. Obejmuje to dostęp do API staging, zrzucanie błędów w monitach debugowania, tryb Undercover (który uniemożliwia Claude'owi ujawnienie, że jest sztuczną inteligencją w publicznych projektach open source) oraz narzędzia dostępne wyłącznie wewnętrznie, takie jak ConfigTool i TungstenTool.

Ironia, która umożliwiła ten wyciek, ma charakter strukturalny. Mapy źródłowe istnieją po to, by wypełnić lukę między skompilowanym wynikiem a oryginalnym kodem źródłowym. Zawierają one oryginalne źródło w `sourcesContent`tablicy, niezależnie od tego, co kompilator usunął. Eliminacja martwego kodu chroni plik wykonywalny. Nie chroni mapy. Bun domyślnie generuje mapy źródłowe. Nikt nie skonfigurował tego tak, by się zatrzymało. Nikt nie dodał niczego `*.map`do `.npmignore`. I tak cała wewnętrzna baza kodu została wysłana do npm w pliku JSON o rozmiarze 59,8 MB.

To nie pierwszy raz. Ten sam problem z mapą źródłową pojawił się w lutym 2025 roku, podczas premiery Claude Code. Anthropic po cichu go usunął. Powrócił w wersji 2.1.88, trzynaście miesięcy później. Proces pakowania nigdy nie został naprawiony. Domyślne ustawienia Buna nigdy nie zostały nadpisane. Firma zajmująca się sztuczną inteligencją, posiadająca najlepszy na świecie model językowy, firma, która stworzyła tryb Undercover Mode specjalnie po to, by zapobiegać wyciekom wewnętrznych informacji, została zamknięta z powodu brakującej linii w pliku `.npmignore`.

Lekcja dla twórców jest strukturalna, a nie anegdotyczna. Jeśli dostarczasz produkt agenta, potrzebujesz dwuwarstwowego systemu funkcji. Eliminacja granic bezpieczeństwa w czasie kompilacji (co nigdy nie może istnieć w zewnętrznym artefakcie). Flagi środowiska wykonawczego do kontroli wdrażania (co istnieje, ale jest wyłączone). Oraz potok kompilacji, który weryfikuje, co faktycznie trafia do opublikowanego artefaktu. `npm pack --dry-run`Przed każdą publikacją. Za każdym razem.

Co to oznacza dla debaty „uprząż kontra model”

Internet spędził 31 marca na katalogowaniu funkcji. System zwierzaków Tamagotchi. Nazwy kodowe zwierząt. Wyrażenie regularne frustracji, które wykrywa, kiedy przeklinasz pod adresem sztucznej inteligencji. To jest zabawne. Nie jest ważne.

Naciśnij Enter lub kliknij, aby zobaczyć obraz w pełnym rozmiarze

Ważne są trzy wzorce.

Samonaprawiająca się pętla zapytań, która traktuje niezawodność jako kluczowy produkt, a nie dodatkowy problem. Pętla liczy 46 000 wierszy, ponieważ ścieżki odzyskiwania (kompresja, zwijanie, eskalacja tokenów,

powrót do modelu zapasowego) są bardziej złożone niż ścieżka bezpieczeństwa. Wykonywanie narzędzi jest wsadowe dzięki zabezpieczeniom współbieżności. Odroczone wykrywanie utrzymuje 18 narzędzi poza pamięcią roboczą do momentu, aż będą potrzebne. Sortowanie alfabetyczne stabilizuje pamięć podręczną komunikatów. Każdy szczegół służy temu samemu celowi: użytkownik nigdy nie powinien widzieć awarii.

System obliczeniowy w czasie snu, który inwestuje cykle bezczynności w konsolidację pamięci. Cztery warstwy pamięci (instrukcje, notatki, przywoływanie sesji, konsolidacja snów) tworzą pierwszą architekturę kognitywną wdrożoną w produkcyjnym narzędziu do kodowania. Trzybramkowy wyzwalacz zapobiega zarówno nadmiernemu, jak i niedostatecznemu śnieniu. Podagent snów działa tylko w trybie odczytu. Limit indeksu wynosi 200 wierszy. Projekt jest ograniczony, przemyślany i odporny na warunki produkcyjne.

System eliminacji cech w czasie kompilacji, który wymusza granice bezpieczeństwa na poziomie artefaktu. Dwa poziomy (kompilacja dla bezpieczeństwa, środowisko uruchomieniowe dla wdrożenia) służą różnym celom. Ironia strukturalna polega na tym, że to właśnie system zaprojektowany do ukrywania wewnętrznych funkcji sprawił, że mapa źródłowa była tak cenna w momencie wycieku: wszystko, co kompilator usunął, nadal znajdowało się na mapie.

To nie są funkcje. To prymitywy. Elementy składowe, których ostatecznie będzie potrzebował każdy system agentów produkcyjnych, niezależnie od tego, czy jest on oparty na Claude, GPT, Gemini, czy modelu open source. Wyciekłe źródło nie ujawniło przewagi konkurencyjnej Claude Code. Ujawniło natomiast wymagania inżynieryjne dla całej kategorii.

Konkurencja ma już architekturę referencyjną. Jeden z deweloperów przeniósł podstawowe wzorce do Pythona przed wschodem słońca. Zespół stworzył 12-sesyjny program inżynierii wstecznej. Porty Rusta są już w toku. Wzorce zostaną przeniesione. Zawsze tak się dzieje.

Znajomość architektury to jednak nie to samo, co jej wykonanie. Claude Code ma 13 miesięcy doświadczenia w udoskonalaniu tych wzorców w produkcji. Ścieżki odzyskiwania dopracowano w 12 wersjach. Reguły uprawnień dostrojone do rzeczywistych obciążeń korporacyjnych. System uprawnień z pięcioma trybami publicznymi, klasyfikatorem uczenia maszynowego do automatycznego zatwierdzania oraz hakami przed wykonaniem, które mogą dyskretnie modyfikować parametry narzędzia przed wykonaniem (model nie wie, że jego dane wejściowe zostały zmienione). Kod źródłowy to migawka. Wiedza operacyjna nie znajduje się w repozytorium.

Pętla samoodniesienia z 31 marca zamyka dyskusję. Deweloperzy zapłacili za model, aby zrozumieć uprząż. Uprząż sprawia, że model jest użyteczny. Model sprawia, że uprząż jest opłacalna. To koło zamachowe nie jest czymś, co można skopiować z mapy źródłowej. To produkt.