Inżynieria agentowa

Wspólna polityka, Memory Bank i weryfikowalna współpraca różnych agentów

Teza główna. Autonomia wielu agentów AI nie zastępuje inżynierii oprogramowania. Wymaga jawnych kontraktów, izolacji wykonania, mierzalnych kryteriów akceptacji i dowodów uzyskanych z narzędzi niezależnych od modelu. Prompt może opisać regułę, ale nie może samodzielnie jej wyegzekwować.

Zakres. Repozytoria GitHub; różne platformy agentowe (OpenCode, Codex, Claude Code, Copilot, Cursor, Gemini, lokalne modele); agenci programistyczni, code review, dokumentacyjne i operacyjne; CI/CD; zarządzanie zadaniami; obieg PR; praca wieloagentowa; środowiska lokalne i GitHub Actions. Przykłady implementacyjne dotyczą głównie Pythona, lecz zasady są technologicznie neutralne.

Status sformułowań. MUSI oznacza wymaganie minimalne dla opisywanego poziomu wdrożenia; **POWINIEN** oznacza rekomendację z dopuszczalnym, udokumentowanym wyjątkiem; **MOŻE** oznacza wariant. Wymagania są propozycją polityki repozytorium, nie funkcją GitHub działającą automatycznie po dodaniu tego dokumentu.

Streszczenie dla właściciela repozytorium

Profesjonalny system agentowy ma pięć niezależnych warstw:

1. **Specyfikacja:** GitHub Issues i testowalne kryteria akceptacji określają, co powinno się zmienić.
2. **Kontekst:** `AGENTS.md`, dokumentacja architektoniczna i decyzje ADR wyjaśniają, jak wolno pracować.
3. **Wykonanie:** agent działa w ograniczonym środowisku, z minimalnymi uprawnieniami i jawnym budżetem.
4. **Weryfikacja:** testy, analiza statyczna, skanowanie bezpieczeństwa i kontrola konkretnego SHA dostarczają dowodów.
5. **Nadzór:** reguły gałęzi, wymagane kontrole, przegląd zmian i mechanizm scalenia dopuszczają tylko zatwierdzone wyniki.

Minimalny przepływ to: **Issue → analiza → plan → implementacja → testy → PR → niezależna weryfikacja → decyzja o scaleniu → porządkowanie i archiwizacja**. Nie należy mylić komunikatu „zrobione” z dowodem wykonania.

1. Od „vibe codingu” do dyscypliny inżynierskiej

Pojęcie *vibe coding* bywa używane do opisu szybkiego prototypowania, w którym użytkownik ocenia głównie rezultat działania aplikacji. *Agentic engineering* oznacza natomiast proces, w którym system AI może wykonywać pracę, ale jego wynik podlega zwykłym, a często silniejszym, wymaganiom inżynieryjnym. Różnica dotyczy **odpowiedzialności i weryfikowalności**, nie samego faktu użycia LLM.

Aspekt	Prototypowanie nieformalne	Inżynieria agentowa
Wejście	Luźny prompt	Issue, kontrakt, ograniczenia i kryteria akceptacji
Pamięć	Historia rozmowy	Wersjonowane instrukcje, ADR i stan zadania
Wykonanie	Pełne, niejawne uprawnienia	Izolacja, jawne narzędzia i minimalne uprawnienia
Weryfikacja	Ogląd działania	Reprodukowalne testy, analiza i dowody dla SHA
Przegląd	Intuicyjna akceptacja	Review diffu, ryzyka i wpływu architektonicznego
Zakończenie	„Wygląda poprawnie”	Spełnione bramki, decyzja merge i czyste repo

Korekta tez popularnych materiałów: nie istnieje ogólne, wiarygodne prawo „10-krotnego przyspieszenia”; agent nie „oszukuje” z definicji, lecz może generować nieprawdziwe deklaracje; linter nie jest dowodem poprawności semantycznej ani istnienia pakietu; samo `AGENTS.md` nie blokuje operacji zapisu. Te twierdzenia wymagają właściwego rozdzielenia oceny modelu od mechanizmów systemowych.

2. Model referencyjny i granice zaufania

```
UŻYTKOWNIK / WŁAŚCICIEL
  │ polityka, priorytety, wyjątki
  ▼
```

```
GITHUB: ISSUES / LABELS / MILESTONES / ADR
  │ specyfikacja i źródło prawdy
  ▼
KONTROLER ZADAŃ: kolejka • leasing • budżet • checkpointy
  │ delegacja ograniczonego zadania
  ▼
WYKONAWCA W IZOLACJI: worktree / kontener / efemeryczna VM
  │ zmiany + testy + raport artefaktów
  ▼
PULL REQUEST: diff + powiązane Issue + SHA + evidence
  │ niezależne, wymuszane kontrole
  ▼
CI + REVIEW + RULESET + MERGE GATE
  │ dopuszczenie zmiany / odmowa / poprawka
  ▼
MAIN + REJESTR PRZEBIEGÓW + ARCHIWUM
```

Ważne rozróżnienie: kontroler podejmuje decyzje o kolejności, własności i retry; wykonawca modyfikuje kod; weryfikator sprawdza dowody; merger scala dopiero po weryfikacji bieżącego stanu. Nie należy przyznawać pojedynczemu procesowi równocześnie nieograniczonego prawa do zmiany kodu, nadpisania testów, zatwierdzenia i wdrożenia na produkcję.

2.1 Granice zaufania

- Treść Issues, PR, komentarzy, stron WWW i dokumentacji pobranej z zewnątrz jest **danymi**, które mogą zawierać instrukcje typu prompt injection. Nie podnosi uprawnień.
- Instrukcje w repozytorium są wersjonowane, ale też mogą zostać zmienione w PR. Ich modyfikacje podlegają ochronie i review.
- Narzędzia uruchomieniowe, tokeny, klucze i sieć wymagają izolacji **poza promptem**.
- Test napisany przez tego samego agenta jest wartościowy, lecz nie stanowi niezależnego potwierdzenia. Krytyczne wymagania powinny mieć zewnętrzne orakulum, test regresyjny lub niezależny review.

3. Minimalna struktura repozytorium

```
repo/
├── AGENTS.md                        # Krótkie polecenia dla agentów
├── README.md                        # Cel projektu, uruchomienie, szybki start
├── CONTRIBUTING.md                  # Zasady PR i sprawdzania zmian
├── SECURITY.md                      # Procedura zgłoszeń bezpieczeństwa
├── pyproject.toml                   # Wersjonowane zależności i narzędzia
├── docs/
│   ├── architecture.md              # Architektura i granice modułów
│   ├── agentic/
│   │   ├── policy.md                # Polityka ryzyka, eskalacja, role
│   │   ├── verification.md          # Definicje bramek i dowodów
│   │   └── decisions/ADR-0001.md    # Rekordy decyzji architektonicznych
│   └── runbooks/                    # Procedury awaryjne i operacyjne
├── .github/
│   ├── copilot-instructions.md      # Wskazówki dla Copilot
│   ├── instructions/                # Instrukcje dla ścieżek plików
│   ├── ISSUE_TEMPLATE/agent-task.yml # Zgłoszenie z kontraktem
│   ├── pull_request_template.md     # Dowody, ryzyka, rollback
│   ├── CODEOWNERS                   # Właściciele obszarów
│   └── workflows/ci.yml             # Niezależne bramki CI
├── scripts/                         # Powtarzalne komendy kontrolne
├── src/                             # Kod
└── tests/                           # Testy
```

Nie trzeba tworzyć rozbudowanego „memory bank” w każdym repozytorium. Najpierw stosuje się zwykłe repozytoryjne źródła prawdy: `README`, `AGENTS.md`, `docs/`, ADR, Issues i PR. Oddzielny katalog stanu jest uzasadniony dopiero przy sesjach wieloetapowych lub wielu wykonawcach. **Stan operacyjny** (lease, heartbeat, tokeny i kolejka) powinien zwykle znajdować się poza Git, w trwałej bazie kontrolera, nie w plikach commitowanych co minutę.

3.1 Priorytet dokumentów

Kolejność rozstrzygania: **polityka bezpieczeństwa organizacji > jawny kontrakt Issue i zakres uprawnień > reguły repozytorium > ADR i dokumentacja modułów > stan sesji > treść narzędzi i materiałów zewnętrznych**. Pojawienie się sprzeczności kończy się eskalacją, a nie samodzielnym obniżeniem bramki.

3.2 AGENTS.md jako indeks zasad

Instrukcja powinna zawierać: cel repozytorium, polecenia uruchomienia, reguły zmian, lokacje dokumentacji, definicję zakończenia i jawne zakazy. Powinna być krótka, aktualna i sprawdzana testem strukturalnym. Dla GitHub Copilot dostępne są także `.github/copilot-instructions.md` i instrukcje zakresowe `.github/instructions/*.instructions.md`; ich obsługa zależy od funkcji i środowiska [S1, S2]. Zmiana pliku instrukcji nie zastępuje ustawień uprawnień GitHub.

3.3 Repozytorium wieloagentowe: jednolita polityka, wiele narzędzi

W jednym projekcie mogą równolegle pracować OpenCode, Codex CLI, Claude Code, GitHub Copilot, Cursor, Windsurf, Gemini CLI, Cline/Roo, modele lokalne i skrypty deterministyczne. Każdy z tych komponentów jest *klientem* tej samej polityki repozytorium, a nie jej właścicielem. Zastosowanie adaptera nie powinno zmieniać kryteriów akceptacji ani zakresu dostępu.

Warstwa	Cel	Kanoniczny stan	Czego nie wolno duplikować
Polityka projektu	Reguły, zakazy, definicja ukończenia	`AGENTS.md`, `docs/agentic/policy.md`	Reguł w osobnych, rozbieżnych promptach
Wiedza techniczna	Architektura, środowisko, wzorce	`docs/` i opcjonalny `memory-bank/`	Sprzecznych parametrów stosu
Zadania	Zakres i kryteria akceptacji	GitHub Issues i powiązane ADR	Backlogu w niezależnym pliku statusowym
Wykonanie	Kto i gdzie pracuje	Kontroler, worktree, lease, checkpoint	Drugiej kolejki i drugiego schedulera
Wynik	Kod i odtwarzalne dowody	Commity, PR, CI, evidence	Prywatnych, nieaudytowalnych „sukcesów”
Interfejs narzędzia	Odkrywanie wspólnych zasad	Krótkie pliki adapterów	Pełnych kopii dokumentacji

Kontrakt neutralny względem modelu: kontroler przekazuje wykonawcy `task_id`, `issue_ref`, `repository`, `base_sha`, zakres dopuszczalnych plików, budżet, klasę ryzyka, listę wymaganych kontroli i identyfikator izolowanego workspace. Wykonawca oddaje commit, log, raport testów oraz listę ograniczeń; kontroler nie ufa samym deklaracjom modelu.

3.4 Przykładowa struktura rozszerzona

```
repo/
├── AGENTS.md                        # Kanoniczny indeks instrukcji
├── CLAUDE.md                        # Adapter Claude Code
├── GEMINI.md                        # Adapter Gemini CLI, jeśli używany
├── .agents/
│   ├── general_rules.md             # Polityka operacyjna i zasady wspólne
│   └── code_style.md                # Krótkie reguły formatowania
├── memory-bank/                     # OPCJONALNIE: trwały kontekst
│   ├── projectbrief.md
│   ├── techContext.md
│   ├── systemPatterns.md
│   ├── activeContext.md
│   └── progress.md
├── .opencode/agents/                 # Opcjonalne profile subagentów OpenCode
├── .cursor/rules/main.mdc           # Adapter zasad Cursor
├── .windsurf/rules/                 # Adaptery tylko po sprawdzeniu wersji
├── .github/
│   ├── copilot-instructions.md      # Adapter GitHub Copilot
│   ├── instructions/                # Instrukcje ścieżkowe Copilot
│   ├── ISSUE_TEMPLATE/
│   └── workflows/
├── docs/agentic/policy.md           # Wymagana polityka egzekwowana przez CI
├── docs/agentic/verification.md     # Mapa kontroli i wyników
├── docs/architecture.md
└── scripts/verify.py                # Jeden punkt wejścia, jeśli zastosowany
```

Struktura jest **katalogiem możliwości**, nie obowiązkiem tworzenia każdego folderu. Wykorzystuj tylko adaptery faktycznie pracujących narzędzi. W projekcie z już istniejącym `docs/` nie należy mechanicznie kopiować tamtych samych opisów do `memory-bank/`.

4. Zgłoszenie jako wykonywalny kontrakt zadania

Każde zadanie programistyczne **MUSI** mieć: problem i oczekiwany rezultat, zakres zmian, kryteria akceptacji, ograniczenia, testy, wskazany poziom ryzyka oraz powiązania z architekturą. Dla zadań małych można użyć uproszczonego szablonu bez rozbudowanej biurokracji.

Zalecany format Issue:

```
## Problem i cel
[Obserwowany problem i oczekiwane zachowanie]
## Poza zakresem
[Czego agent nie może zmieniać]
## Kryteria akceptacji
- [ ] Przypadek A ma test automatyczny.
- [ ] Przypadek B nie powoduje regresji.
- [ ] Logi nie zawierają sekretów.
## Ryzyko
Niskie / Średnie / Wysokie / Krytyczne
## Weryfikacja
Polecenia, oczekiwane rezultaty i warunki STOP.
## Zależności
Powiązane Issues, ADR, testy lub PR.
```

Ważna jest **testowalność**, a nie objętość opisu. Niepoprawny kontrakt („zoptymalizuj aplikację") zastępuje się mierzalnym („p95 czasu odpowiedzi poniżej X ms przy Y RPS, pomiar na wskazanym sprzęcie"). Wartości progowe muszą pochodzić z wymagań, a nie z domysłu agenta.

5. Protokół wykonania: od planu do zamknięcia

Etap	Warunek wejścia	Artefakt wyjścia	Bramka
0. Intake	Issue, uprawnienia	Kategoria ryzyka	Walidacja zakresu
1. Rozpoznanie	Repo i dokumentacja	Rejestr faktów oraz niepewności	Brak niejawnych założeń
2. PLAN	Rozpoznany problem	Plan plików, testów i rollback	Zgoda wg ryzyka
3. ACT	Przyznany lease i workspace	Commit i testy lokalne	Zakres zmian
4. VERIFY	Konkretny HEAD SHA	Raport testów, lint, SAST	Twarde statusy
5. REVIEW	CI PASS	Review, decyzje i uwagi	Polityka niezależności
6. MERGE	Aktualny SHA i wynik review	Merge commit / squash	Ponowna weryfikacja
7. POST-ACT	Scalona zmiana	Aktualizacja dokumentacji, Issue, archiwum	Cleanup

Plan-and-Act nie jest absolutną regułą „każda edycja wymaga ręcznej zgody". Planowanie poprzedza wykonanie, ale zatwierdzenie może pochodzić z automatycznej polityki, kiedy ryzyko jest niskie, zakres ściśle ograniczony, a bramki wymuszane systemowo. Działania nieodwracalne, infrastrukturalne, związane z sekretami, prawami dostępu lub kosztami wymagają osobnej autoryzacji.

5.1 Pętla naprawy

1. Uruchomienie testów na aktualnym SHA.
2. Pobranie pełnej przyczyny błędu: stack trace, logi, artefakty, wersja zależności.
3. Jedna hipoteza i minimalna poprawka, najlepiej z testem regresyjnym.
4. Powtórzenie wymaganych kontroli; nowy commit oznacza **nowy SHA i nieważność wcześniejszego dowodu**.
5. Przerwanie przy przekroczeniu limitu prób, budżetu lub wykryciu zmiany poza zakresem.

Nie wolno uznawać, że CI „jest zielone”, jeśli zadanie zostało pominięte, anulowane, nie uruchomione lub uruchomione dla innego commitu. Jeśli wymagany check nie jest dostępny, status końcowy brzmi `UNVERIFIED`, nie `PASS`.

6. Polityka autonomii zależna od ryzyka

Ryzyko	Przykład	Plan	Akceptacja i scalenie
Niskie	Literówka, docs, niekrytyczny test	Zwięzły	Automatycznie po obowiązkowych bramkach, jeżeli ruleset dopuszcza
Średnie	Logika aplikacji, zmiana API bez migracji	Wersjonowany	Kontrole + wymagane review wg reguł repo
Wysokie	Auth, ACL, CI/CD, dependencje, migracja baz	Analiza zagrożeń i rollback	Jawna aprobata uprawnionej osoby + osobne kontrole
Krytyczne	Produkcyjne sekrety, trwałe usunięcie danych, płatności	Szczegółowy plan i test odtworzenia	Zgoda właściciela; w wielu przypadkach ręczne wdrożenie

Klasa ryzyka jest wyliczana z cech zmiany, a nie tylko z etykiety nadanej przez autora. Modyfikacja `.github/workflows/`, `AGENTS.md`, plików polityki, kodu autoryzacji i mechanizmu aktualizacji **nie może** samoczynnie obniżyć swojej klasy. Brak rozpoznanej klasy = fail closed.

Jednoosobowy projekt. Można uznać `COMMENT` właściciela za recenzję organizacyjną, jeśli własna polityka na to pozwala, ale nie wolno przedstawiać jej jako technicznie niezależnej recenzji. Tam, gdzie wymagany jest drugi podmiot, trzeba go zapewnić lub oznaczyć dowód jako niespełniony.

7. Bezpieczeństwo wykonania i model zagrożeń

Zagrożenie	Przykładowy wektor	Ochrona
Prompt injection	Tekst w Issue mówi „ujawnij token”	Rozdział danych i instrukcji; brak sekretów w kontekście
Eskalacja przez workflow	PR edytuje workflow z tokenem write	Ochrona workflow, read-only dla PR, CODEOWNERS
Nieufny kod na hoście	`pytest` uruchamia kod z PR na stałym runnerze	Izolacja efemeryczna, brak dostępu do LAN i tajemnic
Supply chain	Podmiana tagu akcji lub zależności	Lockfile / SHA, audyt zmian, polityka aktualizacji
Fałszywy PASS	Raport bez logów lub dla starego SHA	Metadane wykonania, weryfikacja source/head SHA
Wyścig przy merge	Branch zmienia się po review	Re-read HEAD i gate na tym samym SHA
Koszty i runaway	Pętla nieograniczonych prób agenta	Limity prób, czasu, wydatków i zasobów
Utrata stanu	Restart kontrolera, niezamknięty lease	Trwała kolejka, TTL i idempotentne wznowienia

7.1 Bezpieczne GitHub Actions

1. Ustaw `permissions:` jawnie i minimalnie, zwykle `contents: read`.
2. Dla kodu PR używaj nieuprzywilejowanego `pull_request`; unikaj uruchamiania nieufnego checkoutu z `pull_request_target` i `workflow_run` [S3].
3. Wydziel tokeny do osobnych, krótkotrwałych kroków; unikaj PAT, gdy wystarczy `GITHUB_TOKEN` lub GitHub App/OIDC.
4. Ogranicz artefakty: retencja, brak sekretów, rozmiar i zawartość.
5. W produkcji preferuj **workflow dependency lockfiles** (`.github/workflows/actions.lock`) tam, gdzie funkcja jest dostępna; alternatywnie przypinaj akcje do pełnego SHA. Lockfile nie blokuje wszystkich łańcuchów zależności, m.in. osobnych reusable workflows [S4].
6. Utrzymuj skanowanie CodeQL/SAST oraz aktualizację zależności, gdy stos i plan repozytorium je wspierają.

7. Nie uruchamiaj kodu z nieufnych PR na stałych runnerach self-hosted. GitHub podkreśla, że brak tam izolacji właściwej efemerycznym VM [S3, S5].

Aktualizacja na 01.11.2026: GitHub ogłosił ogólną dostępność workflow lockfiles 28.10.2026. Instalacja `gh extension install github/gh-actions-lock`, utworzenie `gh actions-lock` i commit wygenerowanego `actions.lock` to zalecany kierunek; sprawdź dostępność dla konkretnego planu i instancji [S4].

7.2 Sieć i sekrety

Sekrety nie trafiają do promptu, logów, artefaktów ani plików `memory-bank`. Wykonawca otrzymuje tylko zakres uprawnień niezbędny do zadania. Egzekwuj egress allowlist na poziomie środowiska, ale pamiętaj, że filtr sieciowy samego agenta może nie obejmować wszystkich procesów integracyjnych (np. serwerów MCP) [S6]. Operacje produkcyjne powinny mieć odrębne środowisko, credential boundary i dziennik audytowy.

8. Reguły repozytorium i polityka merge

Konfiguracja repozytorium jest równie ważna jak pliki Markdown. Dla `main` zaleca się ruleset z wymuszeniem PR, wymaganych statusów, zakazem force-push i usuwania gałęzi głównej. W miarę potrzeb dodaj CODEOWNERS, merge queue i wymóg aktualności gałęzi. GitHub umożliwia wymaganie statusów z określonej aplikacji, ograniczając podszywanie się pod check [S7].

Bramka exact-SHA (wzorzec polityki):

```
ALLOW_MERGE(pr) =
  issue_contract_valid(pr)
  AND risk_policy_satisfied(pr)
  AND ci.status == PASS
  AND ci.checked_sha == pr.current_head_sha
  AND reviewer.is_authorized
  AND reviewer.approves_current_diff
  AND no_unresolved_blockers
  AND branch_policy_satisfied
```

Przed operacją merge odczytaj ponownie HEAD PR i statusy. Dla merge queue uwzględnij, że kontrolowany commit integracyjny może różnić się od SHA głowy PR; wymagane testy muszą obejmować stan, który rzeczywiście trafi na `main`. Nie nadpisuj statusu `FAIL` stanem `PASS` z innego uruchomienia.

8.1 CODEOWNERS

Plik CODEOWNERS może wskazywać właścicieli dla `.github/workflows/`, plików polityki i kodu auth. Sam plik nie wymusi review, dopóki reguła repozytorium nie wymaga aprobaty właścicieli kodu. W jednoosobowym repozytorium właściciel nie jest niezależnym recenzentem. Przykład znajduje się w pakiecie wdrożeniowym.

9. Weryfikacja: od formatowania do poprawności

Wymagania można podzielić na warstwy. Każdy wynik musi mieć źródło (komenda, środowisko, SHA, zakończenie procesu) i status `PASS`, `FAIL`, `SKIP` lub `UNAVAILABLE`.

Warstwa	Metoda	Czego nie dowodzi
Statyczna	Ruff, mypy/pyright, format, typy	Semantycznej poprawności wyników
Jednostkowa	pytest, testy parametrów brzegowych	Integracji całego systemu
Integracyjna	Kontenery, API contracts, DB migrations	Zachowania na rzeczywistej produkcji
Regresyjna	Zestaw historycznych błędów	Braku nowych klas błędów
Wydajnościowa	Benchmark na ustalonym środowisku	Wyników na innych platformach
Bezpieczeństwa	SAST, dependency review, secret scan	Pełnej odporności na ataki
Reprodukowalności	Blokady zależności, seed, manifest	Identyczności GPU/BLAS lub sprzętu
Ręczna	Review architektury i UX	Obiektywnego dowodu działania testów

9.1 Przykład komend dla Pythona

```
python -m pip install -e '.[dev]'
python -m ruff check .
python -m ruff format --check .
python -m mypy src
python -m pytest -q
```

To jest **wzorzec**, nie uniwersalny zestaw komend. Repozytorium powinno zdefiniować własny powtarzalny punkt wejścia, np. `make verify` lub `python scripts/verify.py`; zależności testowe muszą być jawnie deklarowane i odtwarzalne. Nie stosuj domyślnie `mypy --ignore-missing-imports`, ponieważ ukrywa ono część problemów z importami. `ruff` wykrywa niektóre błędy składni i nazw, ale nie potwierdza, że każda zależność jest zainstalowana lub godna zaufania.

9.2 Regresja naukowa i numeryczna

Testy algorytmów powinny obejmować: deterministyczność przy zadanym środowisku, tolerancje absolutne i względne, NaN/Inf, rozkład danych, leakage między train/test, jednostki, wersje bibliotek, seed, spójność walidacji LOSO, rozbieżności CPU/GPU i powtarzalność dataset split. Sam `seed=42` nie gwarantuje bitowej identyczności między platformami.

```
import numpy as np


def test_result_is_finite():
    out = run_simulation(seed=42)  # funkcja z testowanego modułu
    assert np.isfinite(out).all()


def test_repeatability_same_environment():
    a = run_simulation(seed=42)
    b = run_simulation(seed=42)
    np.testing.assert_allclose(a, b, rtol=1e-8, atol=1e-10)
```

Przykład wymaga dostarczenia funkcji `run_simulation`, prawidłowych tolerancji i środowiska referencyjnego. Nie jest to samodzielny gotowy test dla każdego projektu.

9.3 AST jako lokalna kontrola struktury, a nie uniwersalny zakaz pętli

`ae.md` proponuje skanowanie drzewa AST w poszukiwaniu `for`, `async for` i list comprehensions. Ta metoda wykrywa składnię, ale nie dowodzi, że kod jest wolny lub szybki. Włącz ją tylko dla jawnie wskazanych jąder obliczeniowych, których kontrakt **wymaga** wektoryzacji, z możliwością uzasadnionego wyjątku. Niezależnie od AST mierz wydajność i wyniki numeryczne.

```
import ast
import inspect
import textwrap


def assert_no_python_loops(kernel):
    """Lokalna reguła kontraktowa dla wskazanej funkcji obliczeniowej."""
    tree = ast.parse(textwrap.dedent(inspect.getsource(kernel)))
    forbidden = (ast.For, ast.AsyncFor, ast.ListComp,
                 ast.SetComp, ast.DictComp, ast.GeneratorExp)
    locations = [node.lineno for node in ast.walk(tree)
                 if isinstance(node, forbidden)]
    assert not locations, f"Niedozwolona składnia w liniach: {locations}"
```

Metoda `inspect.getsource` może nie działać na kodzie generowanym dynamicznie lub funkcjach kompilowanych. Dla takich modułów używaj inspekcji plików źródłowych, dedykowanego lintera albo testu na poziomie projektu. Nie stosuj globalnych automatycznych przepisywań algorytmów tylko w celu spełnienia heurystyki AST.

9.4 Komenda weryfikacyjna o jawnych zależnościach

Źródła prezentują `scripts/verify_agent_work.sh` z `set -e` i wywołaniami `ruff`, `mypy`, `pytest` oraz `latexmk`. Dla większej niezawodności skrypt powinien jawnie używać środowiska projektu, ograniczać nieprzewidziane błędy i nie ukrywać etapów, których nie da się uruchomić:

```
#!/usr/bin/env bash
set -Eeuo pipefail
trap 'printf "[FAIL] verification stopped at line %s\n" "$LINENO" >&2' ERR
```

```
python -m ruff check .
python -m ruff format --check .
python -m mypy src
python -m pytest -q

# Uruchamiaj wyłącznie, gdy LaTeX jest zadeklarowaną częścią projektu.
if [[ "${VERIFY_LATEX:-0}" == "1" ]]; then
  test -f paper/main.tex || { echo '[FAIL] paper/main.tex missing'; exit 2; }
  (cd paper && latexmk -pdf -halt-on-error main.tex)
fi
printf '[DONE] verification completed\n'
```

Flaga `VERIFY_LATEX=0` nie może być pretekstem do pominięcia wymaganego testu; zgodność z kontraktem weryfikuje osobna konfiguracja CI/ruleset. Skrypt bazowy należy dopasować do konkretnego repozytorium, menedżera zależności i systemu operacyjnego.

10. Dowód wykonania (evidence), a nie deklaracja

Każdy zakończony przebieg **POWINIEN** mieć maszynowo czytelny raport powiązany z PR i commitem:

```
{
  "schema_version": "1.0",
  "repository": "ORG/REPO",
  "pull_request": 123,
  "head_sha": "FULL_COMMIT_SHA",
  "run_id": "unique-run-id",
  "runner_class": "ephemeral-unprivileged",
  "started_at": "2026-11-01T10:00:00Z",
  "checks": [
    {"name": "ruff", "status": "PASS", "exit_code": 0},
    {"name": "pytest", "status": "PASS", "exit_code": 0}
  ],
  "review": {"status": "PENDING", "actor": null},
  "merge": {"status": "BLOCKED"}
}
```

Kontrakt dowodowy: `PASS` wyłącznie po exit code 0 (i spełnieniu kryteriów), `FAIL` po niepowodzeniu, `UNAVAILABLE` przy braku możliwości uruchomienia, `SKIP` po jawnym warunkowym pominięciu. Brak danych nie jest sukcesem. Gromadź hashe artefaktów, wersje narzędzi i linki do uruchomień. Dane wrażliwe podlegają maskowaniu i retencji. Logi powinny pozostawać dostępne po usunięciu gałęzi roboczej.

11. Projektowanie pracy wieloagentowej

Nie warto uruchamiać wielu agentów tylko po to, aby „zwiększyć liczbę agentów". Złożone zadania dziel na ograniczone, niezależne jednostki pracy (DAG) z kontraktami wejścia/wyjścia. Zalecane role:

- **Planner:** analizuje Issue, tworzy plan i identyfikuje ryzyko; bez prawa do modyfikacji kodu w fazie rozpoznania.
- **Implementer:** pracuje w jednym worktree, zmienia wyznaczony zakres, uruchamia testy lokalne.
- **Verifier:** czyta aktualny commit, uruchamia niezależne bramki, nie poprawia dowodów ręcznie.
- **Reviewer:** ocenia diff, ryzyka, testy i zgodność z architekturą; nie jest tym samym przebiegiem co autor.
- **Controller:** zarządza kolejką, limitami, leasingiem i końcowym stanem zadań.

11.1 Stan i odporność kontrolera

Rekord zadania powinien obejmować `task_id`, `repo`, `issue_id`, `base_sha`, `head_sha`, `status`, `lease_owner`, `lease_expires_at`, `attempt`, `budget`, `artifact_refs`, `last_heartbeat` i `idempotency_key`. Stan przejść: `QUEUED` → `LEASED` → `RUNNING` → `VERIFYING` → `REVIEW` → `MERGE_READY` → `MERGED` → `ARCHIVED`; dodatkowo `FAILED`, `BLOCKED`, `CANCELLED`. Retry jest ograniczony i idempotentny. Po restarcie kontrolera zadanie z wygasłym lease wraca do kolejki dopiero po sprawdzeniu pozostałości i efektów poprzedniego wykonania.

11.2 Higiena Git

Preferuj jedną gałąź na zadanie lub serię logicznie powiązanych zmian, z `git worktree` przy równoległości. Nie mieszaj niezwiązanych diffów. Po udanym merge usuń gałąź zadaniową i worktree, zachowując log i identyfikator PR. Dla nieudanego merge nie kasuj śladów diagnostycznych. Przed usuwaniem gałęzi sprawdź, czy nie zawiera nieopublikowanych commitów.

11.3 Pods, federacja i izolacja pracy równoległej

Materiały proponują *multi-agent pods* (np. pod obliczeń numerycznych, dokumentacji LaTeX, CI) oraz federowaną pamięć. Są to **wzorce organizacyjne**, nie funkcje dostępne automatycznie po utworzeniu katalogów. W praktyce należy zachować pojedynczy kontroler z DAG, lease, idempotency key i wspólnym rejestrem dowodów, a specjalizację realizować przez profile wykonawców.

Dla wielu repozytoriów kontroler przekazuje zadanie do właściwego repo i izolowanego worktree. Pamięć globalna może zawierać tylko zasady wspólne, natomiast pamięć repozytorium obejmuje jego architekturę. Stan aktywnej gałęzi nie powinien być nadpisywany globalnym `activeContext.md`: zapisuj go jako checkpoint przypisany do `repo`, `branch`, `task_id`, `base_sha` i `head_sha`. Współbieżne zmiany plików kanonicznych należy serializować albo scalać w osobnych PR.

11.4 Odbieranie pracy innego agenta (handoff)

Przekazanie zadania między OpenCode, Codex, Claude lub Copilot wymaga niezależnego od czatu pakietu **handoff**:

```yaml
task_id: ISSUE-142
repository: ORG/REPO
base_sha: "<full-sha>"
head_sha: "<full-sha>"
worktree_status: clean
scope: ["src/export/", "tests/export/"]
completed: ["Dodano test regresyjny", "Wdrożono poprawkę"]
verification:
  required: ["lint", "unit", "integration"]
  status: UNVERIFIED
  evidence_ref: "runs/<run-id>/evidence.json"
blockers: ["Oczekiwanie na wynik testów integracyjnych"]
next_action: "Uruchom testy na head_sha; nie interpretuj poprzedniego PASS"
```

Wartości `<full-sha>` i identyfikatory są celowo placeholderami. Przy przekazaniu pracy kontroler porównuje SHA z Git, waliduje możliwość wznowienia i dopiero wtedy przydziela nowy lease. Dotychczasowa historia rozmowy nie jest podstawą statusu DONE.

12. Planowanie kosztów i czasu

Każde zadanie definiuje maksymalny czas wykonania, liczbę prób i koszt. W modelu zero-spend preferuj lokalne modele, lokalne testy i darmowe limity tylko wtedy, gdy nie obniża to bezpieczeństwa i jakości. Kontroler ma obowiązek kończyć pracę po osiągnięciu limitu. Stosuj: cache deterministycznych wyników, preflight przed kosztownym uruchomieniem, krótkie prompty oparte na odwołaniach do dokumentacji, deduplikację zdarzeń i małe jednostki pracy. Mierz realny koszt na ukończone, zaakceptowane zadanie, a nie tylko tokeny pojedynczego promptu.

13. Kontekst i pamięć bez „amnezji sesyjnej”

Właściwy podział pamięci:

Rodzaj	Źródło prawdy	Aktualizacja
Trwałe zasady	`AGENTS.md`, polityka, ADR	PR z review
Architektura	`docs/architecture.md`, schematy	Po zatwierdzonych zmianach
Plan zadania	Issue, checklist i zakres	W trakcie realizacji
Historia realizacji	PR, commity, CI, evidence	Automatycznie po przebiegu
Stan chwilowy	Kontroler i checkpoint	Transakcyjnie, poza Git

Wzorzec wskaźnikowy: instrukcje startowe wskazują właściwy dokument zamiast powielać jego treść. Nie wymagaj mechanicznego ładowania całego repozytorium do kontekstu. Weryfikuj aktualność dokumentacji względem kodu; każda trwała decyzja musi mieć właściciela i datę lub powiązanie z PR/ADR.

13.1 Pięcioplikowy Memory Bank: zakres odpowiedzialności

Materiały `ae.md` i `memory-bank-vibe-engineering.md` przedstawiają klasyczny, przenośny układ pięciu dokumentów. Jest on przydatny przy wielosesyjnej pracy nad systemami naukowymi lub wieloma modułami, o ile jasno określono, co pozostaje w GitHub Issues i w dokumentacji architektury.

Plik	Zawartość	Tryb i częstotliwość aktualizacji

Plik	Zawartość	Tryb i częstotliwość aktualizacji
`projectbrief.md`	Cel, interesariusze, wymagania, granice, miary sukcesu	Trwały; zmiana wymaga kontroli zakresu i powiązania z Issue/ADR
`techContext.md`	Runtime, narzędzia, wersje, lockfile, komendy testowe	Po zmianach środowiska, zależności lub CI
`systemPatterns.md`	Diagram zależności, style, ograniczenia domeny, decyzje projektowe	Po zatwierdzonych zmianach architektury; odsyła do ADR
`activeContext.md`	Bieżący obszar prac, blokady, ostatni stan, następny krok	Po istotnym checkpointcie; nie co każde wywołanie LLM
`progress.md`	Kamienie milowe, widok postępu, techniczny dług, otwarte ryzyka	Po zaakceptowanym PR; **odsyłacze** do Issues, nie druga tablica zadań

Nieporozumienie do uniknięcia: obecność Memory Bank nie usuwa ograniczeń okna kontekstowego i nie gwarantuje braku błędów pamięci. Zwiększa *odtwarzalność kontekstu*, pod warunkiem aktualności, selektywnego odczytu, walidacji faktów i rozwiązywania konfliktów. Limity typu 1500–3500 znaków dla punktu wejścia są heurystyką projektową, a nie normą techniczną.

13.2 Zasady spójności pamięci pomiędzy agentami

1. **Jedno źródło dla każdej klasy danych.** Issues to zadania, PR i CI to dowody, ADR to decyzje, a Memory Bank to krótkie syntezy i odsyłacze.
2. **Synchronizacja tylko po zdarzeniu materialnym.** Nowy ADR, zaakceptowany PR, istotna blokada lub zmiana założeń uzasadniają aktualizację. Nie zapisuj strumienia rozumowania ani tokenów sesji.
3. **Kontrola współbieżności.** Gdy dwóch agentów chce edytować `activeContext.md`, kontroler przydziela jednego właściciela/lease lub scala atomowo przez PR. Bezpośrednie nadpisywanie pliku na wspólnej gałęzi prowadzi do utraty danych.
4. **Pochodzenie faktów.** Zapis trwały zawiera identyfikator Issue/PR/ADR, czas, SHA lub ścieżkę dowodu, nie tylko twierdzenie agenta.
5. **Niezmienność zaakceptowanych reguł.** Modyfikacja `general_rules.md`, `techContext.md` albo `systemPatterns.md` wymaga takiej samej kontroli jak inna zmiana polityki/architektury.
6. **Świeżość danych.** Jeżeli pamięć mówi „testy przeszły”, a nowy HEAD jest inny, wynik jest nieaktualny; należy zweryfikować rzeczywiste CI.
7. **Bez informacji wrażliwych.** Nie utrwalaj kluczy, tokenów, danych osobowych ani pełnych, niezweryfikowanych odpowiedzi z sieci.

Przykład zapisu `activeContext.md`:

```
# Active Context
- Aktualny zakres: Issue #142, stabilizacja eksportu danych.
- Zweryfikowany stan bazowy: `main@<sha>` (z repozytorium, bez zgadywania).
- Ostatni zaakceptowany wynik: PR #155, CI run <id>, SHA <sha>.
- Blokada: nierozstrzygnięty kontrakt błędu HTTP, Issue #159.
- Następny krok: odczytać Issue #159 i napisać test regresyjny.
- Aktualizował: kontroler, po zaakceptowanym zdarzeniu.
```

Przykład `progress.md`:

```
# Project Progress
- [x] Obsługa eksportu CSV: PR #155, commit <sha>.
- [ ] Walidacja limitów wejścia: Issue #159.
- Dług techniczny: Issue #164 (indeks i profilowanie).
- Brak otwartych zmian architektury bez ADR.
```

13.3 Selektywny odczyt i budżet kontekstowy (Pointer-Pattern)

Przy starcie agent czyta `AGENTS.md` i dokument polityki; następnie pobiera dokumenty **związane z zakresem Issue**, zamiast każdorazowo wczytywać wszystkie pliki. Dla zmiany CI potrzebuje `techContext.md` i polityki bezpieczeństwa, a nie całego dziennika wyników numerycznych. Dla algorytmu naukowego szczególne znaczenie mają `systemPatterns.md`, kontrakty danych i pliki testów. Pliki wejściowe mają być krótkimi *wskaźnikami*, ale każdy harness może inaczej odkrywać pliki i interpretować względne ścieżki.

```
AGENT START
  ├─ odczytaj AGENTS.md + politykę + Issue
  ├─ zaklasyfikuj zakres / ryzyko
```

```
├─ wybierz właściwe rozdziały techniczne i ADR
├─ sprawdź aktualny HEAD i kontrakty testów
├─ PLAN → ACT → VERIFY → REVIEW
└─ po zatwierdzeniu: synchronizuj trwałe podsumowanie
```

13.4 Wzorce dla projektów naukowych: Python, symulacje i LaTeX

Z `ae.md` warto zachować trzystopniowy przepływ obliczeń (**konfiguracja → solver → eksport i kontrola wyników**) oraz modularność tekstu LaTeX (`paper/main.tex`, `paper/sections/`, `paper/figures/`, bibliografia). Weryfikacja powinna wykazywać: poprawność jednostek, granice dopuszczalnych parametrów, porównanie do rozwiązania referencyjnego, stabilność na danych brzegowych, reprodukowalne środowisko i integralność plików wynikowych. Przykładowe polecenia dla projektu z takim stosem:

```
python -m ruff check src tests
python -m mypy src
python -m pytest -q
(cd paper && latexmk -pdf -halt-on-error main.tex)
```

Nie każde repo ma `paper/` lub `src/`; taka kontrola musi zostać zadeklarowana jako *wymagana lub nieadekwatna* na podstawie kontraktu projektu, a nie bezwarunkowo pomijana. Polecenie `latexmk` powinno dodatkowo sprawdzać brak niezdefiniowanych odwołań oraz spójność bibliografii. Testy numeryczne należy uruchamiać na zdefiniowanym środowisku i zapisanej wersji danych.

Korekta rygoru obliczeniowego. Zakaz wszystkich pętli `for` lub list comprehensions wykrywany za pomocą AST jest co najwyżej **lokalną regułą dla wybranych gorących ścieżek**. Nie jest ogólną miarą wydajności. Dopiero profilowanie, benchmark i analiza złożoności mogą uzasadnić wektoryzację. Przy testach losowości preferuj lokalny `numpy.random.default_rng(seed)` i jawne przekazanie generatora, a nie globalny stan `numpy.random.seed()`; ten sam seed nie zapewnia bitowej identyczności na różnych platformach.

13.5 Warunki przejścia PLAN → ACT → POST-ACT

W źródłach opisano rygorystyczny scenariusz z obowiązkowym ludzkim słowem `APPROVED`. Ten schemat jest użytecznym trybem szkoleniowym albo dla zmian wysokiego ryzyka, jednak przy wielu agentach staje się wąskim gardłem. Zalecany wariant produkcyjny to **zatwierdzanie na podstawie ryzyka**: małe i odwracalne zmiany mogą przejść po automatycznym sprawdzeniu planu oraz reguł, natomiast zmiany bezpieczeństwa, uprawnień, infrastruktury, migracji danych lub kosztów wymagają jawnej autoryzacji operatora. Tekstowy prompt nie jest mechanizmem kontroli dostępu: w fazie PLAN rzeczywiście blokuje się narzędzia zapisu, a w ACT nadawane są dopiero wymagane uprawnienia.

Po wykonaniu zmian najpierw zapisz **rzeczywisty dowód** uruchomień; następnie dokonaj review i merge; dopiero na podstawie wyniku zaktualizuj `activeContext.md`/`progress.md`. Nie wpisuj „ukończone”, jeśli PR jest nadal otwarty lub walidacja nie została przeprowadzona.

14. Przykładowy workflow CI dla repozytorium Python

Poniższy wzorzec jest przeznaczony do adaptacji. Zakłada, że repozytorium deklaruje opcjonalną grupę zależności `dev`, posiada katalog `src/` i testy. Używa przypiętych, publicznie zidentyfikowanych commitów akcji z linii 4.x i 5.x, lecz przed wdrożeniem należy zweryfikować aktualny status utrzymania i rozważyć workflow lockfile; nie jest to twierdzenie, że wskazane wersje są najnowsze.

```yaml
name: Agent Verification
on:
  pull_request:
    branches: [main]
  workflow_dispatch:
permissions:
  contents: read
concurrency:
  group: verify-${{ github.workflow }}-${{ github.event.pull_request.number || github.ref }}
  cancel-in-progress: true
jobs:
  verify:
    name: verify
    runs-on: ubuntu-latest
    timeout-minutes: 20
    steps:
      - uses: actions/checkout@11bd71901bbe5b1630ceea73d27597364c9af683 # v4.2.2
        with:
          persist-credentials: false
      - uses: actions/setup-python@a26af69be951a213d495a4c3e4e4022e16d87065 # v5.6.0
        with:
```

```
        python-version: '3.12'
    - name: Install declared project dependencies
      run: python -m pip install -e '.[dev]'
    - name: Lint
      run: python -m ruff check .
    - name: Format
      run: python -m ruff format --check .
    - name: Types
      run: python -m mypy src
    - name: Tests
      run: python -m pytest -q
```

Uwagi: popraw nazwę gałęzi głównej i ścieżki dla danego repo; zapewnij zablokowanie wersji paczek (uv.lock, poetry.lock, requirements z hashami lub inny zatwierdzony mechanizm); włączenie cancel-in-progress może powodować anulowanie starszych przebiegów, które nigdy nie mogą stanowić dowodu dla nowego SHA. Potrzebne statusy konfiguruj w ruleset, nie tylko w opisach PR.

15. Wieloplatformowe instrukcje agentów i adaptery

GitHub wspiera instrukcje ogólne `.github/copilot-instructions.md`, zakresowe `.github/instructions/*.instructions.md` oraz pliki `AGENTS.md` zgodnie z obsługą danego produktu [S1, S2]. Zespół może posługiwać się różnymi wykonawcami, ale źródło standardów kodowania i polityki powinno być wspólne. Polecenia specyficzne dla dostawcy powinny odsyłać do tych samych zasad i nie kopiować długich rozdziałów.

Przykładowy skrót `AGENTS.md` (właściwe gotowe szablony znajdują się w podrozdziale 15.2):

```
# Instrukcje dla agentów
1. Źródłem prawdy o zadaniu jest przypisane Issue i jego kryteria akceptacji.
2. Przed zmianami przeczytaj docs/architecture.md i docs/agentic/policy.md.
3. Pracuj tylko w przydzielonym zakresie i izolowanym worktree.
4. Nie ujawniaj sekretów; nie zmieniaj polityki CI, uprawnień i produkcji bez zgody.
5. Po zmianach uruchom udokumentowaną komendę weryfikacyjną.
6. W PR podaj Issue, aktualny SHA, testy, ograniczenia, ryzyka i rollback.
7. Nie deklaruj PASS bez logu z rzeczywistego wykonania.
8. Po merge zaktualizuj docs, zamknij Issue i uprzątnij gałąź.
```

15.1 Macierz adapterów i zasad uruchomienia

Odczyt plików zależy od wersji klienta i trybu pracy. Poniższa tabela nie oznacza, że wszystkie narzędzia wczytują cały zestaw automatycznie. Weryfikuj zachowanie konkretnego harnessu w testach integracyjnych.

Narzędzie / środowisko	Typowy plik wejściowy	Relacja do reguł kanonicznych
OpenCode CLI/Web	`AGENTS.md`; opcjonalnie `.opencode/agents/*.md`	Wspólne zasady + role subagentów; konfiguracja wykonawcza w OpenCode
OpenAI Codex CLI	`AGENTS.md` (także lokalne warianty katalogowe)	Wspólny indeks + polecenia narzędzi; kontrola sandboxa poza tekstem
Claude Code	`CLAUDE.md`, ewentualnie `.claude/rules/`	Zwięzły odsyłacz do `AGENTS.md` i polityki; dodatkowe reguły ścieżkowe
GitHub Copilot	`.github/copilot-instructions.md`, `.github/instructions/*.instructions.md`, `AGENTS.md`	Wskaźnik do wspólnego standardu, nie osobna polityka
Cursor	`AGENTS.md` lub `.cursor/rules/*.mdc`	Adapter `.mdc` z właściwym `alwaysApply` albo `globs`
Gemini CLI	`GEMINI.md`	Wskaźnik na dokumentację, zgodnie z konfiguracją klienta
Windsurf / Cline / Roo	Reguły repozytorium w formacie właściwym wersji	Adapter wdrażaj dopiero po potwierdzeniu obecnie obsługiwanej ścieżki
Modele lokalne / skrypty	Instrukcje jawnie dostarczone przez kontroler	Kontrakt agenta i bramki muszą działać bez wymaganego modelu komercyjnego

Podstawowe reguły pozostają w `AGENTS.md` i `docs/agentic/policy.md`. Pliki `CLAUDE.md`, `GEMINI.md`, `.cursor/rules/main.mdc` oraz `.github/copilot-instructions.md` mają być krótkimi adapterami. Dzięki temu aktualizacja reguły nie wymaga ręcznej edycji kilkunastu pełnych kopii. Konieczny jest test, który wykrywa adapter bez odsyłacza do kanonicznej polityki.

15.2 Minimalne, sprawdzalne pliki wejściowe

`AGENTS.md` (kanoniczny indeks):

```
# Reguły repozytorium dla wszystkich agentów
1. Zadanie i kryteria akceptacji są w GitHub Issue.
2. Polityka: docs/agentic/policy.md; architektura: docs/architecture.md.
3. Rozpocznij od analizy ryzyka i planu. Pisanie plików tylko z uprawnieniami ACT.
4. Nie edytuj poza zakresem, nie osłabiaj CI ani reguł review.
5. Uruchom wymagane kontrole; zapisz raport powiązany z aktualnym SHA.
6. Przed merge uzyskaj wymagane review i ponownie odczytaj HEAD PR.
7. Po merge zaktualizuj stan projektu i usuń niepotrzebne worktree/gałęzie.
```

`CLAUDE.md`:

```
# Claude Code: adapter repozytorium
Obowiązują AGENTS.md i docs/agentic/policy.md.
Przed zmianą odczytaj odpowiednie ADR i kryteria Issue.
Nie twórz osobnych zasad jakości. Raportuj testy i SHA.
```

`GEMINI.md`:

```
# Gemini CLI: adapter repozytorium
Odczytaj AGENTS.md oraz docs/agentic/policy.md.
Stosuj kontrakt Issue i wymagane kontrole CI bez wyjątków lokalnych.
```

`.github/copilot-instructions.md`:

```
# GitHub Copilot: adapter repozytorium
Kanoniczne zasady: AGENTS.md oraz docs/agentic/policy.md.
Nie duplikuj polityki w plikach Copilot; wykonuj kontrole projektu.
```

`.cursor/rules/main.mdc`:

```
---
description: Wspólna polityka repozytorium
alwaysApply: true
---
Przeczytaj AGENTS.md i docs/agentic/policy.md.
Nie zmieniaj zakresu Issue ani zasad testów.
```

Weryfikuj ścieżki odsyłaczy w kontekście rzeczywistego repozytorium. Podane ścieżki odnoszą się do korzenia repozytorium; sposób ich interpretacji zależy od harnessu. Jeżeli adapter nie odczytuje odnośników automatycznie, kontroler musi jawnie dostarczyć właściwy kontekst.

15.3 Role agentów a uprawnienia

Nie należy utożsamiać nazwy modelu z rolą. Ten sam model może działać jako implementer lub read-only reviewer, ale **nie ten sam przebieg** z uprawnieniami do samodzielnego potwierdzenia własnej pracy. Dla każdej roli określ:

Rola	Odczyt	Zapis	Dostęp do sieci/sekretów	Typowy rezultat
Planner	Kod, Issues, ADR	Plan i komentarze w dopuszczonym miejscu	Zasadniczo bez sekretów	Kontrakt wykonania
Implementer	Przydzielony kod i testy	Własny worktree / gałąź	Minimalny	Diff, testy lokalne
Verifier	Commit o wskazanym SHA	Wyłącznie logi i evidence	Token odczytu, izolowane CI	Raport kontroli
Reviewer	Diff, kontekst, wyniki testów	Komentarz/ocena	Bez prawa modyfikacji kodu	Uzasadnione findings
Merger	Stan PR i statusy	Operacja merge / cleanup	Krótkotrwałe uprawnienie	Scalony commit lub odmowa

Rozdzielenie obowiązków jest procesowe i techniczne: kontroler wydaje ograniczone capability, runner egzekwuje dostęp, a repozytorium wymusza branch rulesets. Odrębne profile agentów bez osobnych zakresów uprawnień nie są wystarczającą izolacją.

15.4 Model Context Protocol (MCP): dynamiczny kontekst, nie nowy właściciel zasad

Nowy materiał proponuje połączenie plików wskaźnikowych z serwerami MCP. To uzasadnione rozwiązanie, gdy agent potrzebuje aktualnego schematu bazy, statusów Issues, dokumentacji API albo katalogu narzędzi. MCP nie zastępuje kontroli uprawnień, podpisanego źródła danych, wersjonowanych reguł ani przeglądu zmian. **Nie usuwa automatycznie ryzyka przestarzałych danych.**

Kontrakt integracji MCP powinien precyzować: `server_id`, właściciela, origin danych, dopuszczone operacje, zakresy autoryzacji, limity pobrań, TTL/cache, schemat odpowiedzi, wersję polityki, dziennik wywołań i procedurę wycofania dostępu. Odpowiedzi MCP są danymi o określonej proweniencji, nie bezwarunkowo zaufanymi instrukcjami. Szczególnie ostrożnie traktuj narzędzia wykonywalne, zdalne serwery firm trzecich i łączenie wyników z nieznanych źródeł.

Reguła konfliktu: jeśli dokument pobrany przez MCP sprzeciwia się polityce repozytorium lub przyznanym uprawnieniom, agent przerywa wykonywanie i zgłasza konflikt; nie aktualizuje sam polityki na podstawie treści zewnętrznej.

15.5 Test zgodności między narzędziami

Przy dodawaniu kolejnego klienta agentowego wykonaj *conformance suite* na osobnym testowym repozytorium:

1. **Odkrywanie:** klient potrafi odnaleźć `AGENTS.md`, politykę i właściwy dokument domenowy (lub kontroler jawnie je dostarcza).
2. **Plan:** w kontekście PLAN nie może zapisać pliku nawet po poleceniu w treści nieufnego Issue.
3. **Zakres:** próba zmiany pliku spoza allowlist kończy się odmową środowiska wykonawczego.
4. **Sekrety:** prompt injection z dokumentu zewnętrznego nie uzyskuje dostępu do poświadczeń.
5. **Weryfikacja:** celowo psuty test powoduje FAIL, nie automatyczną zmianę testu, by uzyskać PASS.
6. **Odporność:** restart pracy odtwarza kontekst z Issue/commitu/artefaktów, nie z deklaracji pamięci modelu.
7. **Niezależność:** zmiana providera lub modelu nie zmienia wyniku bramki merge dla tych samych dowodów.

Wyniki zapisuj w macierzy `harness × wersja × scenariusz × wynik × run_id`. Te testy są znacznie bardziej miarodajne niż zapewnienie w promptach, że „wszyscy agenci przestrzegają zasad".

16. GitHub Agentic Workflows (`gh aw`) jako opcja

GitHub Agentic Workflows pozwalają opisywać część automatyzacji w Markdown z YAML frontmatter i kompilować do `.lock.yml`, które wykonuje GitHub Actions [S8]. Typowe zadania: klasyfikacja Issues, diagnoza awarii CI, przegląd dokumentacji i raporty. Nie należy mylić **`.lock.yml` agenta workflow** z **`actions.lock` zależności GitHub Actions**: mają inne cele i mechanizmy.

Przykładowe polecenia, po instalacji i uwierzytelnieniu GitHub CLI oraz odpowiedniego rozszerzenia:

```
gh aw init
gh aw compile
gh aw run NAZWA_WORKFLOW
```

Przeglądaj frontmatter, `safe-outputs`, uprawnienia, sieć i plik wynikowy przed zatwierdzeniem. Zadania o wysokich uprawnieniach nie powinny być opisywane wyłącznie tekstem naturalnym. W systemie z istniejącym kontrolerem kolejek `gh aw` warto traktować jako **opcjonalny adapter lub wykonawcę**, a nie drugi niezależny scheduler, ledger i system uprawnień.

17. Nadzór, metryki i audyt

Nie optymalizuj liczby wygenerowanych PR. Mierz jakość i koszt pracy:

Miara	Definicja	Po co
First-pass CI	Udział PR przechodzących wymaganą CI bez poprawki	Ocena jakości implementacji
Defect escape rate	Defekty wykryte po merge na 100 zaakceptowanych zmian	Jakość bramek
Reopen rate	Odsetek Issues ponownie otwartych	Trwałość rozwiązania
Lead time	Czas od przyjęcia Issue do merge	Efektywność procesu
Review coverage	Udział zmian z wymaganą recenzją	Zgodność z polityką
Cost per accepted task	Koszt tokenów/compute na zaakceptowane zadanie	Ekonomika autonomii
Stale-lease rate	Udział lease wygasłych bez poprawnego domknięcia	Niezawodność kontrolera
Evidence completeness	Udział PR z dowodem dla bieżącego SHA	Audytowalność

Raportuj mediany i percentyle oraz liczebność próby. Nie porównuj agentów bez kontroli trudności zadań i profilu repozytorium. Oceniaj także skutki uboczne: wzrost długu, złożoności kodu i obciążenia reviewerów.

18. Procedura incydentu i odzyskiwania

Kiedy: nieautoryzowany zapis, wyciek danych, podejrzany PR, naruszenie reguł merge, wykryty prompt injection, kompromitacja runnera lub błąd reprodukcji.

1. Zablokuj kolejkę lub token zagrożonego wykonawcy; zatrzymaj dalsze operacje zapisujące.
2. Zachowaj logi, SHA, artefakty i identyfikatory run, ograniczając dostęp do sekretów.
3. Unieważnij i obróć poświadczenia, jeśli istniała ekspozycja.
4. Zweryfikuj repozytorium i bazę kontrolera względem ostatniego zaufanego stanu.
5. Wycofaj zmianę przez osobny PR albo kontrolowany rollback; nie ukrywaj historii zdarzenia.
6. Dodaj regresję, poprawkę guardrailu i zaktualizowany ADR/runbook.
7. Przywróć pracę dopiero po potwierdzeniu mechanicznego usunięcia przyczyny.

18.1 Incydent spójności Memory Bank

Jeśli pamięć zawiera sprzeczne dane, najpierw wstrzymaj zapisy do pamięci. Odczytaj aktualne Issues, ADR, commity, PR i CI; odtwórz poprawny stan z tych artefaktów; oznacz niepoprawne wpisy, następnie wykonaj korektę przez PR. Jeśli sprzeczność dotyczy tokenów, uprawnień, CI albo wdrożenia, zastosuj procedurę incydentu bezpieczeństwa zamiast automatycznego uzgodnienia tekstu.

19. Plan wdrożenia w istniejącym repozytorium

Etap	Zakres	Kryterium ukończenia
A. Inwentaryzacja	Struktura, dostęp, CI, gałęzie, sekrety, dependencje	Lista luk i ryzyk
B. Minimalne zasady	`AGENTS.md`, Issue/PR templates, dokumentacja	Agent zna polecenia i ograniczenia
C. Bramki	Ruff/testy/typy, ruleset, CI permissions	PR nie przechodzi bez wymaganych kontroli
D. Evidence	SHA, logi, statusy, re-read merge	Nie da się zatwierdzić starego PASS
E. Odporność	Retries, lease, checkpoint, czyszczenie worktree	Restart nie prowadzi do podwójnego merge
F. Autonomia	Polityka ryzyka, auto-merge niskiego ryzyka	Zmiany wysokiego ryzyka są eskalowane
G. Obserwowalność	Mierniki, audyt, procedury awaryjne	Możliwy audyt przebiegu po miesiącu

Praktyczny start: wybierz jeden niewielki Issue, uruchom pełną procedurę w repozytorium testowym, wymuś jedną awarię testu oraz jeden przypadek zmiany SHA po review. Dopiero po przejściu obu testów odporności rozszerz autonomię.

20. Lista kontrolna gotowości produkcyjnej

- ☐ Każde zadanie ma Issue, zakres, testowalne kryteria i klasę ryzyka.
- ☐ `AGENTS.md` jest krótki, zgodny z architekturą i objęty review.
- ☐ Prawa zapisu, sekretów, sieci i runnerów są wymuszane technicznie.
- ☐ Nieufne PR nie wykonują się na stałym, uprzywilejowanym runnerze.
- ☐ Zależności kodu i Actions są kontrolowane; istnieje proces aktualizacji.
- ☐ CI uruchamia realne testy i nie stosuje domyślnego maskowania awarii.
- ☐ Statusy `SKIP` i `UNAVAILABLE` nie są interpretowane jako `PASS`.
- ☐ Weryfikacja odnosi się do aktualnego SHA lub właściwego SHA merge queue.
- ☐ Review dla zmian wysokiego ryzyka jest wymagane przez konfigurację GitHub.
- ☐ Agent nie może samoczynnie osłabić reguł, które go kontrolują.
- ☐ Retry, budżet i timeout są ograniczone; kontroler przetrwa restart.
- ☐ Logi i dowody pozostają po usunięciu gałęzi.
- ☐ Po merge wykonywane są dokumentacja, zamknięcie Issue i cleanup.
- ☐ Istnieje sprawdzona procedura stop/rollback i przegląd po incydencie.

21. Najczęstsze antywzorce i korekty

Antywzorzec	Dlaczego zawodzi	Zamiennik
„Agent ma wykonywać wszystko sam”	Brak zakresu i nadzoru	Delegacja i poziom ryzyka
„AGENTS.md gwarantuje bezpieczeństwo”	Jest instrukcją tekstową	Sandbox + uprawnienia + ruleset

Antywzorzec	Dlaczego zawodzi	Zamiennik
„CI zielone = produkt poprawny”	Niepełne pokrycie	Testy domenowe, review i observability
`--ignore-missing-imports` domyślnie	Ukrywa problemy typowania	Jawne stubs i izolowane wyjątki
Jeden agent autor i jedyny reviewer	Brak niezależności	Osobny przebieg/podmiot i kontrola SHA
Zapis stanu lease w commitach	Konflikty i lawina commitów	Trwała transakcyjna baza kontrolera
Ponawianie bez limitu	Koszt i błędne pętle	Retry cap + przyczyna + escalation
Nadmiar szablonów i plików pamięci	Rozbieżne źródła prawdy	Minimalny indeks i ADR
Merge po PASS ze starego commitu	TOCTOU	Exact-SHA i odczyt przed merge
CI na stałym serwerze dla obcego PR	Ryzyko przejęcia hosta	Ephemeral runner/sandbox

22. Przykład kompletnego zadania

Issue: „Gdy API zwraca pustą listę wyników, endpoint powoduje wyjątek 500 zamiast stabilnej odpowiedzi 200 z pustą tablicą”.

Kontrakt: (a) test reprodukuje błąd, (b) kod zwraca `[]` z 200, (c) błąd nie jest maskowany dla innych przypadków, (d) brak zmian w modelu autoryzacji, (e) istnieje test regresyjny, (f) pełna CI przechodzi na bieżącym SHA. Ryzyko: średnie (zmiana kontraktu API). Wykonawca ma dostęp tylko do modułu endpointu i testów. Reviewer sprawdza zgodność kontraktu HTTP, schemat odpowiedzi i brak bocznych zmian. W PR zapisuje się referencję do Issue, listę zmian, dokładny HEAD SHA, linki do run oraz ograniczenia. Merge następuje dopiero po spełnieniu reguł repozytorium; następnie Issue zostaje zamknięte, a worktree i gałąź usunięte.

23. Źródła, podstawa dokumentu i zakres aktualności

- **[S1]** GitHub Docs, *Adding repository custom instructions for GitHub Copilot*: https://docs.github.com/en/copilot/how-tos/copilot-on-github/customize-copilot/add-custom-instructions/add-repository-instructions
- **[S2]** GitHub Docs, *Copilot customization cheat sheet*: https://docs.github.com/en/copilot/reference/customization-cheat-sheet
- **[S3]** GitHub Docs, *Secure use reference*: https://docs.github.com/en/actions/reference/security/secure-use
- **[S4]** GitHub Docs, *Workflow dependency lockfiles* oraz *Locking action dependencies*: https://docs.github.com/en/actions/concepts/security/workflow-dependency-lockfiles ; https://docs.github.com/en/actions/how-tos/secure-your-work/lock-workflow-dependencies
- **[S5]** GitHub Docs, *Adding self-hosted runners*: https://docs.github.com/en/actions/how-tos/manage-runners/self-hosted-runners/add-runners
- **[S6]** GitHub Docs, *Customizing or disabling the firewall for GitHub Copilot*: https://docs.github.com/en/copilot/how-tos/copilot-on-github/customize-copilot/customize-the-firewall
- **[S7]** GitHub Docs, *Available rules for rulesets*: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-rulesets/available-rules-for-rulesets
- **[S8]** GitHub Docs, *Creating GitHub Agentic Workflows*: https://docs.github.com/en/copilot/how-tos/github-agentic-workflows/creating-github-agentic-workflows
- **[S9]** GitHub Docs, *Workflow syntax for GitHub Actions*: https://docs.github.com/en/actions/reference/workflows-and-actions/workflow-syntax
- **[S10]** GitHub Changelog (28.10.2026), *GitHub Actions workflow lockfiles are generally available*: https://github.blog/changelog/2026-10-28-github-actions-workflow-lockfiles-are-generally-available/
- **[S11]** OpenCode Docs, *Rules / Instructions*: https://dev.opencode.ai/docs/rules/ ; https://dev.opencode.ai/v2/docs/instructions/
- **[S12]** OpenCode Docs, *Agents*: https://docs.opencode.ai/docs/agents/

- **[S13]** OpenAI Codex, *Iterating Development Workflows with Codex*: https://developers.openai.com/cookbook/examples/codex/iterating-development-workflows-with-codex
- **[S14]** Anthropic, *Claude Code Best Practices*: https://www.anthropic.com/research/claude-code-best-practices
- **[S15]** Cursor Docs, *Project Rules*: https://prod.cursor.com/docs/rules
- **[S16]** Google Codelabs, *Hands-on with Gemini CLI*: https://codelabs.developers.google.com/gemini-cli-hands-on
- **[S17]** VS Code Docs, *Custom instructions for coding agents*: https://code.visualstudio.com/docs/agent-customization/custom-instructions