NARZĘDZIOWNIK TESTERA

AI w automatyzacji testów

Narzędzia, kontekst projektu, workflow i antywzorce AI w automatyzacji testów.

PO CO AI W AUTOMATYZACJI TESTÓW

AI może wspierać testera automatyzującego w całym cyklu pracy z testem: od analizy wymagania i projektowania scenariuszy, przez implementację i refactoring, po uruchamianie testów i analizę niepowodzeń. Największą wartość daje wtedy, gdy korzysta z kontekstu projektu i istniejących wzorców, zamiast tworzyć rozwiązania od zera.

Dzięki temu mniej czasu zajmują powtarzalne zadania, a więcej można poświęcić na jakość architektury testów, diagnozę problemów i ocenę ryzyka.

Zasada nadrzędna

AI może wygenerować kod i zaproponować poprawkę, ale nie rozstrzyga, czy test sprawdza właściwe zachowanie systemu. Każdą zmianę zweryfikuj w kontekście wymagań, standardów projektu i rzeczywistego wyniku testu.

GDZIE AI POMAGA W TESTACH

OBSZAR	CO MOŻE ZROBIĆ AI	CO MUSI SPRAWDZIĆ TESTER
Projektowanie testów	Analizuje wymaganie, endpoint albo fragment kodu i proponuje brakujące scenariusze, przypadki brzegowe, wartości graniczne oraz błędne ścieżki.	Czy scenariusze wynikają z wymagań, kryteriów akceptacji i rzeczywistego ryzyka biznesowego.
Pisanie testów	Tworzy implementację na podstawie istniejących testów, fixture'ów, helperów, Page Objectów i konwencji projektu.	Czy kod nie tworzy równoległych abstrakcji i jest zgodny ze standardem frameworka.
Refactoring	Usuwa duplikację, wydziela helpery, poprawia nazwy i upraszcza kod testów.	Czy zmiana nie osłabia asercji, nie ukrywa problemu i nie zmienia celu testu.
Analiza failów	Łączy test, trace, stack trace, request/response i logi, aby wskazać prawdopodobną przyczynę oraz dowody.	Czy diagnoza jest oparta na źródłach, a nie na domyśle, i czy rozróżnia błąd aplikacji, testu, danych oraz środowiska.
Review	Wskazuje słabe asercje, sztywne waity, kruche lokatory, duplikację, współdzielone dane i zbędny setup przez UI.	Czy rekomendacje poprawiają stabilność i wiarygodność testu, a nie tylko skracają kod.
Rozbudowa automatyzacji	Pomaga rozszerzać istniejący zestaw testów z wykorzystaniem obecnych wzorców i komponentów projektu.	Czy nowe testy pasują do architektury, danych testowych i poziomu testowania przyjętego w projekcie.

JAK DAĆ AI DOBRY KONTEKST

ROZDZIEL KONTEKST NA PROMPTY, ZASADY I WORKFLOW

Nie próbuj zmieścić wszystkiego w jednym prompcie. Stałe zasady przenieś do instrukcji projektu, powtarzalne procesy do workflow lub skilla, a dostęp do zewnętrznych systemów oprzyj na MCP, CLI lub API.

- **Prompt:** Konkretne zadanie wykonywane teraz.
- **Instrukcje projektu:** Stałe zasady, konwencje i ograniczenia niezależnie od treści prompta.
- **Workflow:** Powtarzalny proces wykonywany krok po kroku, np. analiza ticketu albo diagnostyka nieudanego testu.
- **MCP, CLI, API:** Sposób pobierania kontekstu z systemów i wykonywania działań, np. w Jirze, ReportPortalu czy przeglądarce.

ZAPISZ STAŁE ZASADY PROJEKTU

Nie tłumacz agentowi przy każdym zadaniu, jak działa projekt. Stałe zasady zapisz w instrukcjach projektu, np. AGENTS.md, CLAUDE.md, .github/copilot-instructions.md albo .cursor/rules. Obsługiwany format i lokalizacja zależą od używanego narzędzia.

Co warto zapisać:

- Komendy do uruchamiania testów, lintowania i sprawdzania typów
- Struktura projektu i stosowane wzorce
- Zasady tworzenia danych testowych
- Preferowane lokatory
- Pliki i operacje, których agent nie powinien modyfikować

TWÓRZ WORKFLOW ZAMIAST POWTARZAĆ PROMPTY

Jeżeli regularnie wykonujesz z AI ten sam zestaw kroków, zapisz go jako workflow zamiast za każdym razem opisywać

Zamiast prosić o każdy krok osobno

przeczytaj ticket -> sprawdź dokumentację -> znajdź testy -> napisz test -> uruchom -> przeanalizuj błąd

W gotowym workflow podajesz np. numer ticketu, a agent wykonuje kolejne kroki według ustalonego procesu: pobiera wymagania, analizuje repo, przygotowuje testy, uruchamia je i zwraca wynik wraz ze zmianami. Zacznij od małych workflow:

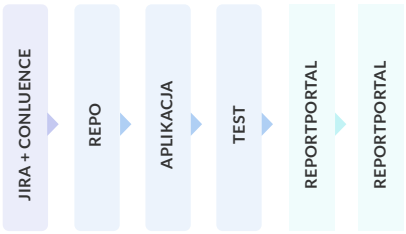
- Jira i Confluence -> propozycja scenariuszy testowych
- Ticket i repozytorium -> implementacja i uruchomienie testów
- ReportPortal, trace i logi -> analiza nieudanego testu

Dopiero potem łącz je w większe procesy.

PODŁĄCZ ŹRÓDŁA ZAMIAST WKLEJAĆ KONTEKST

Jeśli polityka klienta pozwala, nie kopiuj danych ręcznie do prompta. Pozwól agentowi pobierać kontekst bezpośrednio ze źródeł.

- **Jira + Confluence:** Agent może przez MCP albo API pobrać ticket, kryteria akceptacji i powiązaną dokumentację, zanim zacznie przygotowywać albo analizować test.
- **ReportPortal:** Przy analizie faila agent może pobrać launch, wynik testu, stack trace, logi, historię wcześniejszych uruchomień i załączniki.
- **Logi aplikacji:** Jeśli agregator logów udostępnia MCP albo API, agent może pobrać logi z czasu wykonania testu, najlepiej po correlation ID. Może to być np. Elastic, Kibana, Graylog, Splunk albo inne narzędzie używane w projekcie.
- **GitHub:** Przez MCP albo API agent może pobierać kontekst z pull requestów i issues bez ręcznego wklejania go do prompta.



Agent może zbierać kontekst z kilku źródeł

PRACUJ W PRAWDZIWEJ PRZEGLĄDARCE

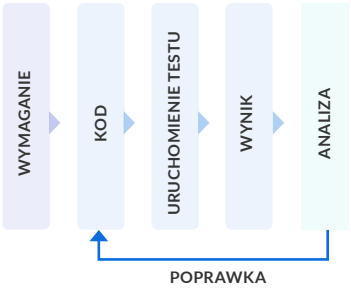
Playwright CLI i Playwright MCP umożliwiają agentowi pracę z rzeczywistą aplikacją. Wybór zależy przede wszystkim od sposobu pracy.

OPCJA	NAJLEPSZE DO	ZALETY	KIEDY UWAŻAĆ
CLI	Pracy z kodem, uruchamiania testów, generowania trace'ów i szybkich operacji w dużym repozytorium.	Jest lekkie, przewidywalne i dobrze pasuje do istniejących komend projektu, CI oraz lokalnego workflow.	Agent musi znać poprawne komendy, katalogi i ograniczenia. Bez instrukcji projektu może uruchamiać zbyt szeroki zakres testów albo działać w złym miejscu.
MCP	Interaktywnej eksploracji aplikacji, pobierania kontekstu z wielu źródeł i pracy agenta z narzędziami zewnętrznymi.	Ułatwia pracę na żywym kontekście: ticketach, dokumentacji, logach, repozytorium, przeglądarce albo ReportPortalu.	Każdy serwer MCP trzeba traktować jak osobne narzędzie z uprawnieniami. Szczególnie ryzykowne jest połączenie odczytu zewnętrznych treści i operacji zapisu.
API	Stabilnych integracji z systemami, pobierania danych według konkretnych parametrów i automatyzacji powtarzalnych procesów.	Daje kontrolowany, jednoznaczny dostęp do danych i dobrze sprawdza się w workflow, które mają działać powtarzalnie.	Wymaga poprawnej autoryzacji, obsługi limitów, błędów i zakresów dostępu. Nie powinno zastępować zgody na przetwarzanie danych.

PRACA Z AI OD WYMAGANIA DO WYNIKU

ZAMKNIJ PĘTLĘ OD WYMAGANIA DO WYNIKU

Nie kończ na samym wygenerowaniu kodu. Agent powinien uruchomić właściwy zakres testów, przeanalizować wynik i sprawdzić, czy test rzeczywiście wykrywa błąd.



Pętla od wymagania do poprawki

Jeśli to możliwe, upewnij się też, że test rzeczywiście przestaje przechodzić po celowym zepsuciu sprawdzanej reguły.

AUTOMATYZUJ TRIAGE FAILI

Dobry kandydat na pierwszy workflow AI

```
fail -> ReportPortal -> trace
  -> correlation ID -> logi aplikacji
  -> klasyfikacja
```

Correlation ID: Identyfikator pozwalający powiązać wykonanie testu z requestami i logami aplikacji. Dzięki niemu agent może szybciej znaleźć zdarzenia dotyczące konkretnego faila.

KROK	CO ROBI AGENT	WYNIK / KONTROLA TESTERA
1	Zbiera dane o failu z raportu, kodu testu, stack trace'u, trace'u i request/response.	Tester sprawdza, czy agent analizuje właściwe uruchomienie i właściwą wersję testu.
2	Łączy wykonanie testu z logami aplikacji, najlepiej po correlation ID.	Wynikiem powinien być zestaw źródeł, a nie ogólna hipoteza bez dowodów.
3	Porównuje obecny fail z wcześniejszymi uruchomieniami i znanymi problemami.	Tester weryfikuje, czy problem jest nowy, powtarzalny czy wygląda na flakiness.
4	Klasyfikuje przyczynę: aplikacja, test, dane, środowisko albo znany problem.	Klasyfikacja musi zawierać uzasadnienie i wskazane dowody.
5	Proponuje następny krok: zgłoszenie błędu, poprawkę testu, zmianę danych, rerun albo eskalację.	Tester zatwierdza kierunek przed zmianą kodu, danych lub konfiguracji.

Celem nie musi być automatyczna naprawa. Sama klasyfikacja problemu i wskazanie dowodów mogą znacząco skrócić triage.

ODDZIEL GENEROWANIE OD REVIEW

Nie pozwól, żeby ten sam agent wymyślił rozwiązanie i sam je zaakceptował. Przy większym workflow warto rozdzielić pracę na etapy:

- Analiza wymagania i scenariuszy
- Implementacja testu
- Review wygenerowanego kodu
- Analiza wyników i failów

Osobny krok review zmniejsza ryzyko, że ta sama analiza i te same założenia przejdą bez dodatkowej kontroli. Może wykonać go człowiek albo drugi agent.

MIERZ EFEKT, NIE UŻYCIE AI

AI nie jest celem. Celem jest szybsze dostarczanie testów bez pogorszenia jakości. Sprawdzaj:

- czas od ticketu do gotowego testu,
- czas analizy faila z AI i bez AI,
- odsetek zmian wymagających dużych poprawek na review,
- trend flaky testów po wdrożeniu AI.

Jeśli czas spada, ale rośnie liczba poprawek lub niestabilnych testów, AI tylko przesuwa pracę na późniejszy etap.

KONTROLA POPRAWNOŚCI TESTU

OPIERAJ OCZEKIWANY WYNIK NA ŹRÓDLE

AI może proponować dane i warianty scenariuszy, ale oczekiwany wynik powinien wynikać z:

- wymagania,
- kryteriów akceptacji,
- kontraktu API,
- albo znanego zachowania systemu.

Jeśli model wymyśli zarówno test, jak i oczekiwany rezultat, może utrwalić błędne założenie.

UWAŻAJ NA SELF-HEALING

AI może znaleźć nowy lokator po zmianie UI, ale automatyczna poprawka testu nie zawsze jest właściwym rozwiązaniem. Zmiana może oznaczać:

- zwykły refactoring UI,
- zmianę wymagania,
- prawdziwą regresję.

NIE POPRAWIAJ PO CICHU

Automatyczna zmiana lokatora powinna trafić do review. Najpierw sprawdź, dlaczego test przestał przechodzić.

NIE MASKUJ PROBLEMU

Agent może szybko sprawić, że test znów będzie zielony. Poniższa tabela pomaga odróżnić realną poprawkę od zmiany, która tylko maskuje problem.

PODEJRZANA ZMIANA	DLACZEGO JEST RYZYKOWNA	CO SPRAWDZIĆ ZAMIAST TEGO
Osłabiona asercja	Test przechodzi, ale może już nie sprawdzać kluczowego zachowania systemu.	Czy asercja nadal wynika z wymagania, kontraktu API albo kryteriów akceptacji.
Dłuższy timeout, wait albo retry	Może ukryć problem z synchronizacją, wydajnością albo niestabilnym środowiskiem.	Czy test czeka na właściwy warunek, zdarzenie lub stan aplikacji.
Nowy lokator po zmianie UI	Zmiana może oznaczać regresję, zmianę wymagania albo niejednoznaczny element.	Czy nowy lokator wskazuje ten sam element i czy zachowanie użytkownika się nie zmieniło.
Akceptacja snapshotu lub baseline'u	Różnica wizualna może zostać utrwalona bez oceny, czy była oczekiwana.	Jaka jest przyczyna różnicy i czy została zaakceptowana przez właściciela funkcji.
Pominięcie testu lub ignorowanie wyjątku	Problem znika z raportu, ale nadal istnieje w produkcie albo środowisku.	Czy istnieje decyzja o czasowym wyłączeniu testu, issue z przyczyną i plan powrotu.
Zmiana danych, setupu albo kodu aplikacji	Agent może naprawić objaw poza zakresem zadania i zmienić warunki testu.	Czy zmiana należy do zakresu pracy testera i czy została zatwierdzona w review.

Cel

Po poprawce test nadal powinien wykrywać regresję, a nie tylko przechodzić.

AUTONOMIA, UPRAWNIENIA I BEZPIECZEŃSTWO

CHECKLISTA PRZED URUCHOMIENIEM AGENTA

✓	CO SPRAWDZIĆ	PO CO
○	**Narzędzie jest dopuszczone**	Upewnij się, że model, agent i ewentualny serwer MCP są zgodne z polityką projektu lub klienta.
○	**Zakres danych jest bezpieczny**	Sprawdź, czy do agenta nie trafią sekrety, tokeny, dane produkcyjne ani informacje objęte ograniczeniami klienta.
○	**Kontekst jest dostępny u źródła**	Podłącz wymagania, repozytorium, dokumentację, logi lub ReportPortal zamiast ręcznie wklejać duże fragmenty kontekstu.
○	**Instrukcje projektu są aktualne**	Zweryfikuj komendy, konwencje, chronione pliki, dane testowe i zasady review, zanim agent zacznie generować zmiany.
○	**Uprawnienia są minimalne**	Nadaj tylko potrzebne dostępy. Do analizy często wystarczy odczyt, a operacje zapisu powinny wymagać kontroli.
○	**Zmiany są odizolowane**	Uruchom agenta na osobnym branchu, w worktree albo sandboxie, szczególnie przy większych zmianach.
○	**Cel zadania jest jednoznaczny**	Określ, czy agent ma analizować, generować kod, uruchomić testy, zrobić review czy tylko przygotować rekomendację.
○	**Granice autonomii są ustalone**	Wskaż, które działania agent może wykonać sam, a które wymagają zatwierdzenia człowieka.

POTWIERDŹ, CZY NARZĘDZIE JEST DOZWOLONE

Każde narzędzie AI, model i serwer MCP używany w projekcie muszą być zgodne z polityką klienta. Nie podpinaj nowych rozwiązań na własną rękę.

Niepotrzebne narzędzie lub uprawnienie

Przed pierwszym użyciem potwierdź dopuszczenie narzędzia, zakres danych i uprawnienia. Informacji przesłanych do niedozwolonego środowiska nie można później „cofnąć".

DOPASUJ AUTONOMIĘ DO RYZYKA

Poziom autonomii dobieraj do ryzyka.

DZIAŁANIE	POZIOM KONTROLI
Odczyt dokumentacji i logów	może być automatyczny
Zmiana kodu	review / PR
Zmiana danych lub środowiska	dodatkowe zatwierdzenie
Operacje produkcyjne (deployment, zmiana danych)	wyraźna autoryzacja

Zasada: Im większy wpływ działania, tym większa powinna być kontrola człowieka.

IZOLACJA PRACY AGENTA

- Git worktree: Osobny katalog roboczy tego samego repozytorium, zwykle na osobnym branchu. Pozwala agentowi pracować nad zmianami bez mieszania ich z bieżącą pracą.
- Sandbox: Może dodatkowo ograniczyć dostęp agenta do plików, sieci i sekretów.

EGZEKWOWANIE ZASAD PRZEZ HOOKI

Hook agenta: Automatyczna akcja uruchamiana przy określonym zdarzeniu, np. po edycji pliku albo przed zakończeniem zadania.

Hook może automatycznie:

- zablokować zmianę chronionych plików,
- uruchomić lint,
- uruchomić typowanie lub testy po zmianie kodu,
- egzekwować zasady projektu.

Nie polegaj tylko na instrukcjach.
Jeśli narzędzie obsługuje hooki, wykorzystaj je do zasad, których nie chcesz zostawiać wyłącznie w dokumentacji.

CHROŃ DANE I UPRAWNIENIA

Nie przekazuj AI sekretów, tokenów, danych klientów ani danych produkcyjnych. Szczególnie uważaj na logi, requesty i pliki, które mogą zawierać takie informacje.

Dostęp agenta
Im większe uprawnienia ma agent, tym większy wpływ może mieć błąd, halucynacja lub niepożądana instrukcja.

ZABEZPIECZ MCP I ZEWNĘTRZNY KONTEKST

Agent nie odróżnia automatycznie danych od instrukcji.

Treści pobrane z ticketów, logów, dokumentacji i MCP mogą zawierać instrukcje wpływające na zachowanie modelu.

Prompt injection: Ticket, dokumentacja, log albo strona mogą zawierać treść, która próbuje zmienić zachowanie modelu lub nakłonić go do niepożądanej akcji.

Dobre praktyki:

- korzystaj tylko z zaufanych serwerów MCP,
- nadawaj im minimalne uprawnienia,
- zachowaj szczególną ostrożność przy połączeniu odczytu zewnętrznych treści z dostępem do zapisu.

Największe ryzyko:

Najbardziej niebezpieczna sytuacja to agent, który jednocześnie:

- czyta zewnętrzne treści,
- może wykonywać operacje zapisu.

NAJCZĘSTSZE ANTYWZORCE

1. **AI napisało, więc merge:** Kod wygenerowany przez AI nadal wymaga review i uruchomienia testów.
2. **Generowanie bez kontekstu:** Bez istniejących testów i zasad projektu AI zacznie tworzyć własne konwencje.
3. **Jeden gigantyczny prompt:** Nie wrzucaj całej dokumentacji, logów i kodu naraz. Niech agent pobiera kontekst wtedy, gdy jest potrzebny.
4. **Wszystko przez MCP:** MCP nie zawsze jest najlepszym wyborem. CLI albo API może być prostsze i zużywać mniej kontekstu.
5. **Pełne uprawnienia:** Do analizy Jiry, Confluence, ReportPortal czy logów często wystarczy dostęp read-only.
6. **Poprawka bez diagnozy:** Przy failu najpierw znajdź przyczynę i dowody, dopiero potem zmieniaj kod.

GOTOWE PROMPTY

Analiza nieudanego testu
Przeanalizuj nieudany test na podstawie kodu testu, trace, stack trace, request/response i logów. Najpierw wskaż najbardziej prawdopodobną przyczynę oraz dowody. Rozdziel problem aplikacji, testu, danych i środowiska. Nie proponuj zmiany kodu, dopóki przyczyna nie zostanie uzasadniona.

Implementacja testu w istniejącym frameworku
Na podstawie wymagania i istniejących testów zaproponuj implementację zgodną z aktualnymi fixture'ami, helperami, Page Objectami, namingiem i sposobem tworzenia danych. Nie twórz nowych abstrakcji, jeśli w projekcie istnieje już rozwiązanie tego samego problemu.

Review testu
Przejrzyj ten test pod kątem jakości i stabilności. Wskaż słabe asercje, sztywne waity, kruche lokatory, współdzielone dane, zbędny setup przez UI, duplikację i testowanie na niewłaściwej warstwie. Dla każdego problemu podaj wpływ i rekomendowaną zmianę.

Złota zasada

ZAPAMIĘTAJ!
Dobra praca z AI nie kończy się na wygenerowaniu testu. Agent powinien korzystać z wymagań i kontekstu projektu, uruchomić test oraz zweryfikować wynik. Ostateczna ocena poprawności pozostaje po stronie testera. z wcześniejszej części rozmowy.