Czat w praktyce: jak rozmawiać, żeby dostać robotę

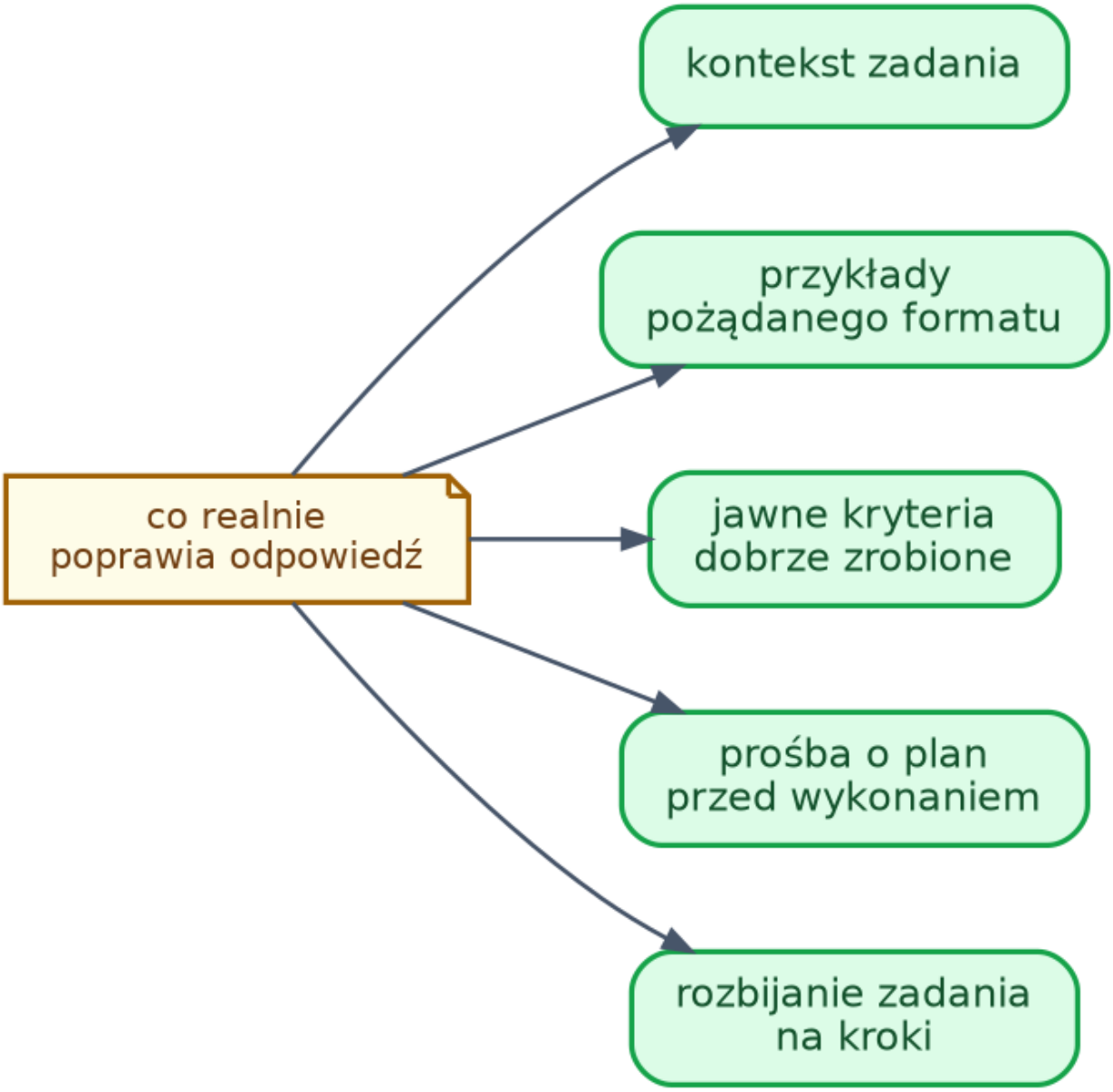
> Codzienna rozmowa z czatem AI ma swoją własną, dość wąską listę rzeczy, które faktycznie podnoszą jakość odpowiedzi — reszta to szum i przyzwyczajenia z rozmowy z człowiekiem, które tu nie działają tak samo. Ta strona zbiera tę listę razem z sygnałami, że rozmowa już się zapętliła.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
AD	Active Directory: katalog Microsoftu z kontami użytkowników, komputerów i grup w firmie
AI	Artificial Intelligence: sztuczna inteligencja, czyli programy naśladujące ludzkie rozumowanie
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu

Co realnie poprawia odpowiedź

Kontekst zadania. Model nie wie, dla kogo piszemy, po co ten tekst powstaje ani jakie ograniczenia obowiązują w naszym środowisku, dopóki mu tego nie powiemy. „Napisz procedurę dołączenia serwera do domeny" i „napisz procedurę dołączenia serwera RHEL10 do domeny AD dla kolegów z zespołu, którzy nie znają realmd" to dwa różne zadania, mimo że różnią się tylko dwoma zdaniami kontekstu. Im więcej milczących założeń zostawiamy modelowi do odgadnięcia, tym większa szansa, że odgadnie źle.



Pięć rzeczy, które realnie podnoszą jakość odpowiedzi czatu (kliknięcie otwiera pełny rozmiar)

Przykłady. Pokazanie trzech do pięciu przykładów pożądanego formatu działa lepiej niż opisanie tego formatu słowami, zwłaszcza gdy chodzi o styl, strukturę albo poziom szczegółowości. Dobry przykład jest różnorodny — nie trzy warianty tego samego przypadku, tylko przypadki pokazujące różne krawędzie tego, czego oczekujemy.

Jawne kryteria „dobrze zrobione". Prośba bez kryterium sukcesu zostawia modelowi wybór, czym ma się kierować, a ten wybór bywa przypadkowy. „Sprawdź tę konfigurację" nie mówi, czy chodzi o bezpieczeństwo, wydajność, zgodność z naszym standardem nazewnictwa czy wszystko naraz. „Sprawdź tę konfigurację pod kątem otwartych portów, które nie są na naszej liście dozwolonych" daje modelowi coś, wobec czego może się rozliczyć — i nam coś, wobec czego możemy rozliczyć jego odpowiedź. To ten sam mechanizm, który w rozdziale o ewaluacji nazywamy

zestawem kryteriów akceptacji; w rozmowie codziennej działa dokładnie tak samo, tylko nieformalnie.

Proszenie o plan przed wykonaniem. Przy zadaniu, które ma więcej niż dwa, trzy kroki, warto najpierw poprosić o opis planu, zanim padnie polecenie „rób". Plan da się przeczytać w kilkanaście sekund i złapać błędne założenie, zanim model zdąży je rozwinąć w dziesięć stron kodu albo w długą procedurę zbudowaną na złym fundamencie. To tania forma kontroli jakości, o wiele tańsza niż poprawianie gotowego wyniku.

Rozbijanie dużego zadania na kroki. Zamiast jednego polecenia „przygotuj mi całą migrację z systemu A do systemu B" lepiej wychodzi seria mniejszych próśb: najpierw mapowanie pojęć między systemami, potem szkielet skryptu, potem obsługa przypadków brzegowych, na końcu przegląd całości. Krótsze kroki łatwiej zweryfikować pojedynczo, a błąd złapany wcześnie nie zdąży się rozprzestrzenić na kolejne kroki — to ten sam mechanizm kumulacji błędów, który dla agentów autonomicznych opisaliśmy na stronie Różnica pomiędzy czatem a agentem.

Czego nie robić

Kolejnego „popraw to" w nieskończoność. Trzecia, czwarta, piąta poprawka tej samej odpowiedzi w tym samym wątku zwykle pogarsza sprawę zamiast ją poprawiać — kontekst puchnie o kolejne błędne wersje, model zaczyna się gubić w tym, która poprawka jest aktualna, a które ustalenie z dziesięciu wiadomości wstecz wciąż obowiązuje. Jeżeli druga poprawka nie trafiła w sedno, zwykle szybciej i taniej wychodzi nowy wątek z jasno spisanym, poprawionym opisem zadania niż kontynuowanie starego.

Wklejanie całego repozytorium albo całego logu. Intencja jest zrozumiała — niech ma pełny obraz — ale efekt bywa odwrotny do zamierzonego: istotny fragment ginie w setkach linii szumu, a okno kontekstowe wypełnia się treścią, która nigdy nie będzie potrzebna do odpowiedzi. Lepiej wkleić przefiltrowany wycinek i dopowiedzieć, gdzie szukać reszty, niż liczyć, że model sam odsieje ziarno od plew z dwudziestu tysięcy linii. Więcej o tym, dlaczego długi, zaszumiony kontekst szkodzi jakości, na stronie Kontekst i pamięć: co model wie, a czego nie.

Pytania bez kryterium sukcesu. „Zrób to ładnie", „popraw ten skrypt" bez powiedzenia, co dokładnie w nim przeszkadza, zostawiają model z zadaniem odgadnięcia naszych oczekiwań. Konkretne „ten skrypt ma działać bez interakcji z klawiatury i zostawiać log w `/var/log`" daje coś sprawdzalnego.

Kiedy nowy czat, a kiedy kontynuacja

Kontynuacja ma sens, dopóki historia rozmowy jest zasobem, a nie balastem — czyli dopóki wcześniejsze ustalenia są wciąż aktualne i potrzebne do kolejnego kroku tego samego zadania. Poprawianie tego samego skryptu krok po kroku, dopytywanie o szczegóły tej samej procedury, rozwijanie tego samego dokumentu — to naturalne kontynuacje.

Nowy wątek ma sens w trzech sytuacjach: zmienia się temat i stara historia jest po prostu nieistotna; rozmowa zapętliła się na tej samej poprawce po raz trzeci czy czwarty; albo model przyjął błędne założenie kilka wiadomości temu i to założenie ciągle wraca, mimo prób sprostowania — w takim wypadku łatwiej zacząć od zera z jasno spisanym, poprawnym opisem niż próbować wyplątać się z błędu utrwalonego w kontekście. Ten trzeci przypadek ma nazwę w literaturze o agentach — zatruwanie kontekstu — i wracamy do niego szerzej na stronie Kontekst i pamięć: co model wie, a czego nie.

Jak czytać odpowiedź krytycznie

Ton pewności siebie w odpowiedzi modelu nie jest sygnałem prawdziwości — model brzmi tak samo pewnie, cytując prawdziwą komendę i wymyśloną. Kilka rzeczy warto sprawdzać zawsze, niezależnie od tego, jak dobrze odpowiedź wygląda:

Czy przywołane fakty — nazwy parametrów, przełączników CLI, nazwy funkcji, numery wersji oprogramowania — istnieją naprawdę, sprawdzone w oficjalnej dokumentacji, a nie tylko brzmią prawdopodobnie.

Czy liczby się zgadzają arytmetycznie, jeżeli odpowiedź zawiera jakiekolwiek wyliczenia — sumy, procenty, konwersje jednostek.

Czy założenia o naszym środowisku, które model przyjął milcząco, są prawdziwe dla naszego konkretnego przypadku, a nie tylko dla ogólnego, podręcznikowego scenariusza.

Czy odpowiedź faktycznie odnosi się do kryterium sukcesu, które podaliśmy na starcie, czy tylko brzmi na temat.

Fiszka

> **TYLKO ODCZYT**
>
> Szkielet dobrego promptu zadaniowego

```
Cel: OpisCeluZadania

Kontekst: OpisSrodowiskaIOgraniczen

Kryteria akceptacji:
- KryteriumPierwsze
- KryteriumDrugie

Format wyniku: OczekiwanyFormatOdpowiedzi

Czego unikać: OgraniczeniaNegatywne
```

Wypełniamy tylko te pola, które mają treść — pusty „Format wyniku" można pominąć, ale „Kryteria akceptacji" warto uzupełniać niemal zawsze, bo to one dają nam później podstawę do oceny, czy odpowiedź faktycznie spełnia zadanie, a nie tylko wygląda na spełniającą.

Przykład:

```
Cel: przygotowanie procedury podłączenia RHEL10 do domeny AD

Kontekst: zespół zna Linuxa, nie zna realmd; domena to ad.zielona.example,
kontroler domeny wskazywany przez DNS

Kryteria akceptacji:
- procedura działa bez ręcznej edycji /etc/hosts
- każdy krok ma komendę do skopiowania i jedno zdanie po co
- osobna sekcja "co sprawdzić po zmianach"

Format wyniku: strona markdown w formacie naszej wiki

Czego unikać: zakładania SSSD skonfigurowanego wcześniej
```

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

> Ola po raz piąty w tym samym wątku pisze do czatu „popraw jeszcze to", poprawiając od rana skrypt migracyjny. Model zaczyna się gubić, która wersja funkcji jest aktualna, i miesza fragmenty z drugiej i czwartej poprawki. Zamiast szóstej prośby o poprawkę zakładamy nowy wątek z jednym, jasno spisanym zdaniem o tym, co dokładnie nie działa. Poprawiony skrypt wychodzi już z pierwszego podejścia w nowej rozmowie.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Kolega wysyła do czatu polecenie „napraw ten skrypt", wklejając pięćset linii kodu bez komentarza, co konkretnie nie działa. Jakich dwóch elementów z listy „co realnie poprawia odpowiedź" tu brakuje?

Odpowiedź

Brakuje kontekstu zadania (co skrypt ma robić, co robi zamiast tego) i jawnego kryterium „dobrze zrobione" (na czym polega „naprawiony" — brak błędu składni, poprawne działanie na konkretnym wejściu, zgodność ze stylem zespołu). Bez tego model może naprawić coś, co wcale nie było problemem.

2. Piąta z rzędu prośba „jeszcze popraw" w tym samym wątku daje odpowiedź gorszą niż druga poprawka. Co zrobić zamiast szóstej prośby o poprawkę?

Odpowiedź

Zacząć nowy wątek z jasno spisanym, poprawionym opisem zadania — najlepiej wykorzystując wnioski z nieudanych poprawek jako część kontekstu w nowym opisie, a nie kontynuować w wątku, którego historia jest już zdominowana przez kolejne, coraz bardziej pogmatwane wersje tej samej odpowiedzi.

Źródła: Prompting best practices, dokumentacja Claude Platform: https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices — Building Effective AI Agents, Anthropic: https://www.anthropic.com/engineering/building-effective-agents

---

Revision #35
Created 2026-09-25 10:33:15 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:09 UTC by Tomasz Konieczny