Dobre praktyki programistyczne

na egzaminach zawodowych INF.03 i INF.04 oraz w biznesie.

Opracowała: Izabela Muskała

Wejherowo, październik 2026

UCZYSZ
www.uczysz.pl
kontakt@uczysz.pl
607 77 11 56

uczysz

Wstęp

Na początku zaznaczę, że nie ma jednej, oficjalnej i zamkniętej liczby dobrych praktyk programistycznych. Programowanie to rzemiosło, które rozwija się od kilkudziesięciu lat, dlatego nowe zasady powstają wraz z rozwojem technologii.

Mogę jednak śmiało powiedzieć, że istnieje kilka kluczowych fundamentów (od kilkunastu do kilkudziesięciu najważniejszych zasad), z których wywodzą się setki mniejszych reguł specyficznych dla konkretnych języków programowania.

Dlaczego to jest ważne: bo stosowanie dobrych praktyk programistycznych pozwala tworzyć kod, który jest łatwy w utrzymaniu, bezpieczny i zrozumiały dla innych.

Oddaję w twoje ręce zestawienie najważniejszych i najczęściej stosowanych dobrych praktyk w świecie IT oraz tych, które są wymagane na egzaminach zawodowych (za nie są punkty!).

uczysz

Dobre praktyki programistyczne stosowane w branży IT

W świecie IT praktyki zostały podzielone na logiczne kategorie:

1. Pisanie czystego kodu (Clean Code)

- **Zasada KISS (Keep It Simple, Stupid):**
  Twórz kod jak najprostszy. Unikaj przekombinowanych rozwiązań, jeśli prosty algorytm spełnia zadanie.
- **Zasada DRY (Don't Repeat Yourself):**
  Unikaj powielania kodu. Jeśli ten sam fragment pojawia się w kilku miejscach, wydziel go do osobnej funkcji lub klasy.
- **Zasada YAGNI (You Aren't Gonna Need It):**
  Nie pisz kodu „na zapas” ani funkcji, których aplikacja obecnie nie potrzebuje.
- **Znaczące nazwy:**
  Nazywaj zmienne, funkcje i klasy tak, aby ich nazwa jasno opisywała ich rolę (np. zamiast p użyj password).

2. Projektowanie architektury (Solidne fundamenty)

- **Zasady SOLID:**
  Zestaw pięciu reguł programowania obiektowego, które zwiększają elastyczność systemu (m.in. zasada jednej odpowiedzialności – każda klasa robi tylko jedną rzecz).
- **Wzorce projektowe:**
  Korzystanie ze sprawdzonych, uniwersalnych szablonów rozwiązań dla powtarzających się problemów (np. Singleton, Fabryka, Obserwator).

3. Kontrola jakości i testowanie

- **Testy automatyczne:**
  Pisanie testów jednostkowych (Unit Tests), integracyjnych i end-to-end (E2E), które gwarantują, że zmiany w kodzie nie popsują istniejących funkcji.
- **Code Review (Przegląd kodu):**
  Sprawdzanie kodu przez innego programistę przed wdrożeniem. Pozwala to wyłapać błędy i dzielić się wiedzą w zespole.
- **Refaktoryzacja:**
  Regularne poprawianie struktury istniejącego kodu bez zmiany jego zewnętrznego działania (czyszczenie "długu technologicznego").

4. Praca z kodem i narzędzia

- **Korzystanie z systemu kontroli wersji (Git):**
  Regularne tworzenie commitów, opisywanie ich jasnymi komunikatami oraz praca na osobną gałęziach (pull requesty).
- **Automatyzacja (CI/CD):**
  Wdrażanie potoków Continuous Integration i Continuous Deployment, które automatycznie budują, testują i wdrażają aplikację.
- **Formatowanie i lintery:**
  Używanie narzędzi automatycznie dbających o jednolity styl kodu w całym zespole (np. Prettier, ESLint).

5. Bezpieczeństwo i dokumentacja

- **Walidacja danych wejściowych:**
  Nigdy nie ufaj danym od użytkownika. Filtruj i sprawdzaj wszystko, co trafia do systemu, aby zapobiec atakom (np. SQL Injection).
- **Aktualna dokumentacja:**
  Utrzymywanie pliku README.md oraz dokumentacji API (np. Swagger), co ułatwia wdrożenie nowych osób do projektu.

uczysz

Dobre praktyki programistyczne na egzaminach INF.03 i INF.04

Na egzaminach zawodowych dla technika informatyka i programisty (INF.03 oraz INF.04), oficjalne kryteria oceniania sprawdzają tylko część komercyjnych dobrych praktyk. Egzaminy te są mocno ustrukturyzowane – klucze odpowiedzi CKE skupiają się przede wszystkim na **czytelności i bezbłędnym działaniu kodu**, a nie na zaawansowanej architekturze.

Oto, jak wymienione wcześniej praktyki przekładają się bezpośrednio na punkty na egzaminie:

1. Praktyki absolutnie WYMAGANE

- **Znaczące nazwy i styl (Clean Code):**
  W oficjalnych arkuszach INF.04 oraz INF.03 istnieją osobne, jawne punkty za to kryterium. W kryteriach oceniania widnieją wprost zapisy:

  „Kod źródłowy zapisano w sposób czytelny: instrukcje w osobnych liniach, stosowane spacje pomiędzy operatorami”.

  „Kod zapisano z wcięciami dla zagnieżdżeń bloków”.

  „Użyto polskie lub angielskie, znaczące nazewnictwo funkcji / metod”.

  Zmienne, funkcje i klasy muszą mieć nazwy mówiące o ich przeznaczeniu. Należy unikać pojedynczych liter jak *a,b,x,* chyba, że to zmienne iteracyjne działające w pętli. Uczeń powinien stosować konsekwentną notację (np. camelCase w Java/JS lub snake_case w Pythonie.

- **Zasada jednej odpowiedzialności (SRP):**
  Jedna funkcja lub metoda powinna wykonywać tylko jedno określone zadanie. Jeśli funkcja ma 100 linii kodu, należy rozbić ją na mniejsze podfunkcje.

- **Zasada DRY (Don't Repeat Yourself):**
  W zadaniach programistycznych na INF.04 regularnie pojawia się wymóg obiektowości lub tworzenia konkretnych metod i funkcji (np. funkcja parsująca, algorytm szyfrujący).

  Jeśli zamiast zamknięcia kodu w reużywalną funkcję uczeń przekopiuje go kilka razy, straci punkty za strukturę aplikacji opisaną w poleceniu.

Pola klas powinny być prywatne (private), a dostęp do nich powinien odbywać się przez publiczne gettery i settery.

Konstruktory powinny służyć wyłącznie do inicjalizacji stanów obiektów.

- **Formatowanie kodu:**
  Brak wcięć (tabulatorów/spacji) oraz pisanie wielu instrukcji w jednej linii gwarantuje utratę punktów w sekcji

  „Rezultat 1: Implementacja i czytelność kodu”.

- **Aktualna dokumentacja:**
  Na egzaminie INF.04 (szczególnie w części z aplikacjami desktopowymi/mobilnymi) jednym z zadań jest przygotowanie dokumentacji aplikacji. Zazwyczaj polega to na zrobieniu zrzutów ekranu, opisie testów lub przygotowaniu plików konfiguracyjnych w odpowiednim folderze (np. dokumentacja/).

2. Praktyki zalecane/wymagane POŚREDNIO

- **Zasada KISS (Keep It Simple):**
  Egzamin trwa tylko 150 minut (**INF.03**) lub 180 minut (**INF.04**). Nie ma czasu na budowanie skomplikowanych struktur. Im prościej i bardziej "pod klucz oceniania" uczeń napisze kod, tym mniejsza szansa na błąd.

- **Obsługa błędów i wyjątków:**
  Kod podatny na błędy (np. wczytywanie pliku testowego, dzielenie przez zmienną wprowadzoną przez użytkownika) należy zawsze umieszczać w blokach try-catch lub try-except. Zapobiega to nagłemu crashowaniu się aplikacji podczas sprawdzania przez egzaminatora.
  Inny sposób (mniej profesjonalny) to zastosowanie instrukcji warunkowych.

  Bardzo często w poleceniach pojawia się warunek typu:

  "Aplikacja uniemożliwia wpisanie wartości ujemnych" lub

  "Pole nie może być puste".

- **Testowanie (ale manualne):**
  Arkusze **INF.04** często wymagają przeprowadzenia testów aplikacji i spisania ich wyników w tabeli testowej (np. dane wejściowe, oczekiwany rezultat, rezultat faktyczny). Uczeń nie pisze tam automatycznych testów jednostkowych, ale testuje kod "z palca".

Do tego celu można wykorzystać Junit – popularny framework służący do tworzenia i uruchamiania powtarzalnych testów jednostkowych lub inne wbudowane w środowisko narzędzia.
Uczeń powinien zrobić testy zarówno przypadków poprawnych jak i skarajnych, np. liczby ujemne, puste ciągi znaków).

- **Dokumentacja**
  Uczeń powinien pisać zwięzłe komentarze dokumentacyjne nad klasami i kluczowymi metodami (opisujące parametry wejściowe i zwracaną wartość)

W INF.03 liczy się semantyka, bezpieczeństwo przesyłania danych i czytelna struktura plików.

1. **Semantyka i struktura HTML:**
   - Używanie tagów semantycznych (<header>, <main>, <nav>, <footer>, <section>) zamiast wszechobecnych .
   - Dodawanie atrybut alt do znaczników <img> oraz pamiętaj o domyślnym kodowaniu <meta charset="UTF-8">.

2. **Separacja kodu (Style i Skrypty):**
   - Nie należy pisać stylów CSS bezpośrednio w pliku HTML (tzw. *inline*). Należy tworzyć osobny arkusz stylów (np. styl.css).
   - Skrypty JavaScript lub PHP należy dołączać w osobnych plikach lub wydzielać je do wyraźnych bloków, dbając o to, by kod JS ładował się na końcu dokumentu (np. przed zamknięciem </body> lub z atrybutem defer).

3. **Bezpieczeństwo baz danych w PHP/SQL:**
   - Należy zawsze filtrować dane z formularzy (używaj mysqli_real_escape_string lub bindowania parametrdziaów w PDO), aby zapobiec atakom SQL Injection.
   - Nie wolno zostawiać otwartych połączeń z bazą – zawsze należy zamykać je na końcu skryptu poprzez mysqli_close($conn).
   - Słowa kluczowe w zapytaniach SQL piszemy wielkimi literami (np. SELECT, WHERE, INSERT INTO), co ułatwia czytanie kodu egzaminatorowi.

4. **Walidacja formularzy:**
   - Należy stosować walidację dwupoziomową – najpierw po stronie klienta (HTML5 np. required, pattern, type="email" oraz JavaScript), a następnie bezwzględnie weryfikować te dane po stronie serwera w PHP.

uczysz

Uniwersalne zasady egzaminacyjne

Praktyka	Dlaczego jest ważna?
Identyczne nazwy plików	CKE w treści zadania podaje dokładne nazwy plików i folderów. Błąd w chociażby jednej literze lub wielkości znaku może skutkować **0 punktów** za dany rezultat, bo skrypt egzaminatorski go nie wykryje.
Formatowanie kodu (Wcięcia)	Bezwzględnie trzeba używać tabulatorów lub spacji. Kod "zbity w jedną linię" utrudnia czytanie i obniża ocenę za kulturę programistyczną.
Brak zbędnego kodu (Dead code)	Należy usuwać niedokończone pętle czy zakomentowane bloki kodu testowego przed zapisaniem ostatecznej wersji na płytę.
Częsty zapis i kopie	Stanowiska egzaminacyjne bywają kapryśne. Bezwzględnie trzeba zapisywać kod co kilka minut (najszybciej wciskając klawisze Ctrl + S)

Podsumowanie:

Podczas egzaminu uczniowie powinni skupić się przede wszystkim na **czytelności (wcięcia, spacje), poprawnym i jasnym nazywaniu zmiennych oraz funkcji** oraz na tym, by program po prostu kompilował się bez błędów. Piękna architektura przegrywa na egzaminie z kodem, który jest prosty, ale w 100% zgodny z treścią zadania.