

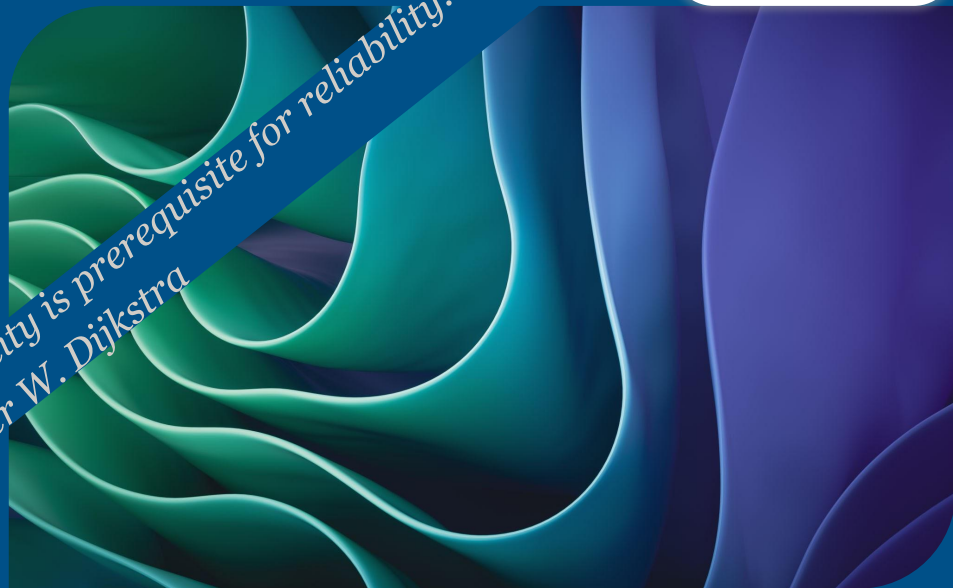
Jakość kodu – Mit, czy Hit?

100%

Michał Kołodziej
Daniel Stasielak

JPWP - projekt
AGH

“Simplicity is prerequisite for reliability.”
Edsger W. Dijkstra



I – Jakość Kodu

Dawn of
The First Day

01 Czym faktycznie jest jakość kodu?

02 Jak rozpoznać takowy kod?

03 Co jeśli się bez niego obejdziemy?

Charakterystyki dobrego kodu

Czytelność

01

Spójność

02

Modularność

03

Niska złożoność

04

Dokumentacja

05

Czytelność

01

Kod powinien być łatwy do zrozumienia, nie tylko dla jego autora, ale dla każdego członka zespołu. Odpowiednio nazwane zmienne i metody, typowanie, czy konsekwentne formatowanie to podstawowe zasady utrzymania czytelnego kodu.

```
1 def howdy(name, age):
2     """no typing"""
3     return f"Howdy {name}, you're {age} years old."
4
5
6 def howdy_typed(name: str, age: int) -> str:
7     """typing"""
8     return f"Howdy {name}, you're {age} years old."
9
10 def get_nth(example_list: List[int], index: int) -> Optional[int]:
11     """typing module example"""
12     if 0 <= index < len(example_list):
13         return example_list[index]
14     return None
```

```
array = [29, 99, 27, 41, 66, 28, 44, 78, 87, 19, 31, 76, 58, 88, 83, 97, 12, 21, 44]
q = lambda l: q([x for x in l[1:] if x <= l[0]]) + [l[0]] + q([x for x in l if x > l[0]]) if l else []
print(q(array))
```

Spójność

02

Stosowanie jednolitych konwencji w całym projekcie, tj. stylu kodowania, nazewnictwa i struktury. W efekcie kod wygląda na napisany przez jedną osobę, nawet jeśli pracuje nad nim cały zespół.

[styleguide](#)

Google Style Guides

Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.

"Style" covers a lot of ground, from "use camelCase for variable names" to "never use global variables" to "never use exceptions." This project ([google/styleguide](#)) links to the style guidelines we use for Google code. If you are modifying a project that originated at Google, you may be pointed to this page to see the style guides that apply to that project.

- [AngularJS Style Guide](#)
- [Common Lisp Style Guide](#)
- [C++ Style Guide](#)
- [C# Style Guide](#)
- [Go Style Guide](#)
- [HTML/CSS Style Guide](#)
- [JavaScript Style Guide](#)
- [Java Style Guide](#)
- [JSON Style Guide](#)
- [Markdown Style Guide](#)
- [Objective-C Style Guide](#)
- [Python Style Guide](#)
- [R Style Guide](#)
- [Shell Style Guide](#)
- [Swift Style Guide](#)
- [TypeScript Style Guide](#)
- [Vim script Style Guide](#)

This project also contains [google-c-style.el](#), an Emacs settings file for Google style.

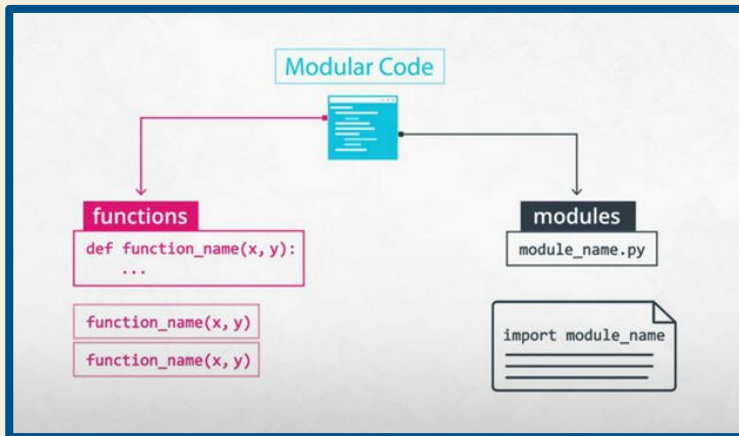
We used to host the cpplint tool, but we stopped making internal updates public. An open source community has forked the project, so users are encouraged to use <https://github.com/cpplint/cpplint> instead.

Modularność

03

Pisząc modularny kod miej na uwadze:

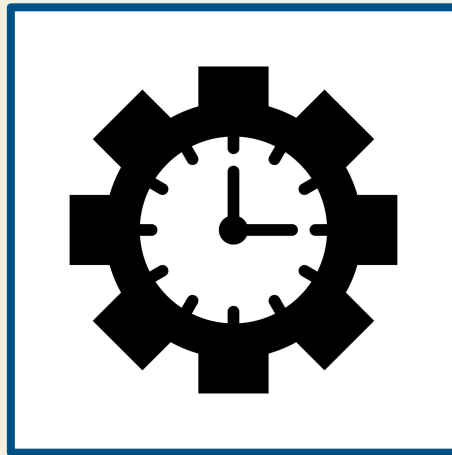
- DRY (Don't Repeat Yourself)
- Dziel duże funkcje na mniejsze, niezależne elementy
- Każda metoda powinna zajmować się jedną rzeczą
- Ogranicz ilość argumentów danej metody



Niska złożoność

04

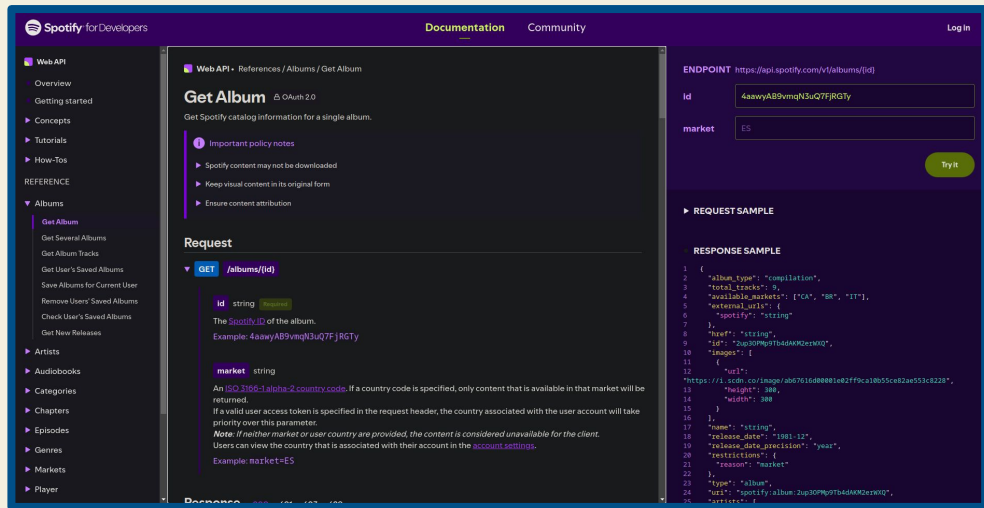
Kod powinien być prosty, bez zbędnych zależności i rozgałęzień. Im mniej wyjątków i skrótów myślowych, tym łatwiej go zrozumieć, przetestować i poprawić.



Dokumentacja

05

Kod powinien być wspierany komentarzami i dokumentacją, tam gdzie to konieczne. Nie jest to wszędzie wymagane, oczywiste sekcje tłumaczą się same, ale bardziej skomplikowane fragmenty kodu powinny być opisane.



The screenshot shows the Spotify for Developers documentation page for the 'Get Album' endpoint. The page is divided into several sections:

- Web API**: Overview, Getting started, Concepts, Tutorials, How-Tos.
- REFERENCE**: Albums (Get Album, Get Several Albums, Get Album Tracks, Get User's Saved Albums, Save Albums for Current User, Remove User's Saved Albums, Check User's Saved Albums, Get New Releases), Artists, Audiobooks, Categories, Chapters, Episodes, Genres, Markets, Player.
- Get Album**: References / Albums / Get Album. Get Spotify catalog information for a single album.
- Important policy notes**: Spotify content may not be downloaded, Keep visual content in its original form, Ensure content attribution.
- Request**: GET /albums/{id}. The {id} is the Spotify ID of the album. Example: 4aawyAB9vmqN3uQ7fjRGTy. The market parameter is optional. Example: market=ES.
- Endpoint**: https://api.spotify.com/v1/albums/{id}. Fields: id, market.
- Request Sample**: curl -X GET https://api.spotify.com/v1/albums/4aawyAB9vmqN3uQ7fjRGTy -H 'Authorization: Bearer <token>' -H 'Accept: application/json'.
- Response Sample**: A JSON object containing album details like album_type, total_tracks, available_markets, external_urls, spotify, href, id, images, name, release_date, release_date_precision, restrictions, and type.

Różne systemy zasad

SOLID

GRASP

YAGNI

DRY

KISS

PLK

II – Testowanie Kodu

**Dawn of
The Second Day**

Podstawowe rodzaje testów

01 Testy jednostkowe

02 Testy integracyjne

03 Testy funkcyjne

04 Testy systemowe

05 Testy regresyjne

Testy Jednostkowe

01

Co testujemy

Pojedyncze funkcje
lub metody w izolacji.

Cel

Upewnić się, że
każdy fragment kodu
działa poprawnie
samodzielnie.

Narzędzia

- pytest
- unittest
- JUnit
- NUnit

Testy Jednostkowe

01 Przykład

```
2 def divide(a, b):
3     if b == 0:
4         raise ValueError("Cannot divide by zero")
5     return a / b
6
7 def test_divide_normal():
8     assert divide(10, 2) == 5
9
10 def test_divide_by_zero():
11     with pytest.raises(ValueError, match="divide by zero"):
12         divide(5, 0)
```

Testy Integracyjne

02

Co testujemy

Współpracę między modułami lub komponentami.

Cel

Sprawdzenie, czy części systemu poprawnie się ze sobą komunikują.

Przykłady

- Serwis + baza danych
- API + logika

Testy Funkcjonalne

03

Co testujemy

Konkretne funkcje systemu zgodnie z wymaganiami.

Cel

Potwierdzić, że system robi to, czego oczekuje użytkownik.

Narzędzia

- Selenium
- Cypress

Testy Systemowe

04

Co testujemy

Cały przepływ działania systemu z punktu widzenia użytkownika.

Cel

Sprawdzić, czy wszystkie warstwy (frontend, backend, baza) działają razem.

Narzędzia

- Cypress
- Playwright
- Selenium

Testy Regresyjne

05

Co testujemy

Czy nowe zmiany nie zepsuły istniejącej funkcjonalności.

Cel

Zapobieganie błędom po aktualizacjach.

To tyle?

- Testy wydajnościowe
- Stress-testy
- Recovery
- Testy dymne
- Testy bezpieczeństwa
- Testy eksploracyjne

III – Praktyka

Dawn of
The Final Day

CI/CD

Continuous Integration

Regularna integracja kodu do wspólnego repozytorium. Każda integracja jest automatycznie testowana, aby szybko wykryć błędy.

Continuous Delivery / Deployment

Po przejściu przez CI, możemy przejść przez

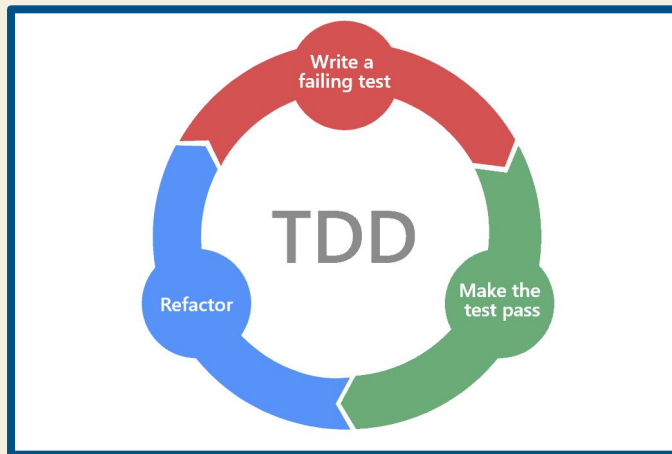
- Cont. Delivery, gdzie zmiany są automatycznie przygotowane do wdrożenia, ale czekają na manualną interwencję
- Cont. Deployment, gdzie wdrożenie również jest automatyczne

TDD - Test Driven Development

Dla TDD wpierw piszemy testy, a potem kod, który musi nie tylko spełniać nasze wymagania i specyfikacje, ale i przejść wszystkie kody.

Charakteryzuje go cykl:

- **RED** – piszemy test, który nie przechodzi
- **GREEN** – piszemy kod, który przechodzi wcześniej napisany test
- **REFACTOR** – wpierw potencjalnie skomplikowaną, zagmatwaną implementację naprawiamy, ale musi ona nadal zdawać test



ATDD - Acceptance Test Driven Development

Zgodnie z filozofią ATDD przed implementacją funkcji pisane są testy akceptacyjne.

Tworzone wspólnie przez:

- Programistów
- Testerów
- Product Ownera / klienta

Skupia się na kryteriach akceptacji, np. kiedy każda strona twierdzi, że dana implementacja jest gotowa.

Użytkownik
podaje
poprawne
dane



Użytkownik
zostaje
zalogowany

Użytkownik
podaje
niepoprawne
dane



Wyświetla
się
komunikat o
błędzie.

BDD – Behaviour Driven Development

Filozofia BDD skupia się na funkcjonalności z perspektywy użytkownika

Zamiast testów pisane są scenariusze w języku naturalnym (np. Język Gherkin: Given–When–Then).

Dzięki BDD kod jest łatwy do pojęcia nawet dla osób nie będących programistami. Pozwala to na prowadzenie znacznie prostszego dialogu z inwestorami, czy managerami nietechnicznymi.

```
Feature: Guess the word
```

```
# The first example has two steps
```

```
Scenario: Maker starts a game
```

```
    When the Maker starts a game
```

```
    Then the Maker waits for a Breaker to join
```

```
# The second example has three steps
```

```
Scenario: Breaker joins a game
```

```
    Given the Maker has started a game with the word "silky"
```

```
    When the Breaker joins the Maker's game
```

```
    Then the Breaker must guess a word with 5 characters
```


sauce

<https://dev.to/prxtikk/how-to-write-clean-and-modular-code-1d87>

https://en.wikipedia.org/wiki/Unit_testing

https://en.wikipedia.org/wiki/Recovery_testing

<https://www.ibm.com/think/topics/system-testing>

www.redhat.com/en/topics/devops/what-is-ci-cd

<https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd>

<https://cucumber.io/docs/gherkin/>

<https://cucumber.io/docs/bdd/>

LLM-y Gemini i ChatGPT

