

Refaktoring kódu

Refaktoring je systematický proces vylepšování designu kódu bez přidávání nových funkcí. Cílem je učinit kód čistším (Clean Code), čitelnějším a snadněji udržitelným. Jak řekl Martin Fowler: „*Jakýkoli blbec může napsat kód, kterému rozumí počítač. Dobří programátoři píší kód, kterému rozumí lidé.*“

1. Kdy je refaktoring potřeba? (Code Smells)

Příznaky, že kód vyžaduje refaktoring, se v komunitě označují jako „**Code Smells**“ (zápachy kódu). Mezi nejčastější patří:

- **Duplicitní kód:** Stejná logika se vyskytuje na více místech (porušení principu DRY – Don't Repeat Yourself).
- **Příliš dlouhá metoda:** Metoda dělá příliš mnoho věcí najednou (porušení [SRP](#)).
- **Velká třída:** Třída, která se snaží být „Božským objektem“ (God Object) a spravuje příliš mnoho agendy.
- **Příliš mnoho parametrů:** Metoda vyžaduje dlouhý seznam argumentů, což ztěžuje její použití a testování.

—

2. Základní techniky refaktoringu

Existují desítky popsaných technik, zde jsou ty nejpoužívanější:

Extract Method (Extrakce metody)

Část kódu z dlouhé metody se vyjme a vytvoří se z ní nová, samostatná metoda s výstižným názvem.

- **Výhoda:** Zvyšuje čitelnost a umožňuje znovupoužití logiky.

Rename Variable/Method (Přejmenování)

Změna názvu proměnné nebo metody tak, aby jasně vyjadřovala svůj účel.

- **Příklad:** Změna `int d;` na `int daysSinceCreation;`

Inline Method (Vložení metody)

Opak extrakce. Pokud je tělo metody stejně jasné jako její název, metoda se zruší a kód se vloží přímo tam, kde se volala.

Move Method (Přesun metody)

Pokud metoda více komunikuje s jinou třídou než s tou, ve které se nachází, přesune se do té druhé.

—

3. Zlaté pravidlo: Testování

Refaktoring by se **nikdy** neměl provádět bez sady automatizovaných testů (Unit testů).

1. Máte funkční kód s testy, které procházejí (Green).
 2. Provedete malou úpravu (Refaktoring).
 3. Spustíte testy. Pokud stále procházejí, vnější chování zůstalo zachováno.
 4. Pokud testy selžou, vrátíte změnu zpět.
-

4. Přínosy refaktoringu

- **Snížení technického dluhu:** Kód se nestává „neudržitelným monstrem“.
 - **Snazší hledání chyb:** V přehledném kódu se chyby schovávají mnohem hůře.
 - **Rychlejší vývoj v budoucnu:** Čistá architektura umožňuje rychlejší přidávání nových funkcí.
 - **Lepší porozumění:** Programátor při refaktoringu do hloubky pochopí, jak kód funguje.
-

5. Refaktoring vs. Optimalizace

Je důležité tyto pojmy nezaměňovat:

- **Refaktoring:** Cílem je **čitelnost a design**. Může se stát, že kód bude po refaktoringu o milisekundu pomalejší, ale mnohem čistší.
- **Optimalizace:** Cílem je **výkon** (rychlost, využití paměti). Často může kód znepřehlednit v zájmu efektivity.

Související články:

- [Principy SOLID](#)
- [Unit Testing a kvalita kódu](#)
- [Základy OOP](#)

Tagy: *programming software-engineering refactoring clean-code maintenance*

From:

<https://www.serviceit.cz/> - **IT ENCYKLOPEDIE**

Permanent link:

<https://www.serviceit.cz/doku.php?id=it:sw:refactoring>

Last update: **2026/01/02 18:54**

