Lernnachweis C1G, C1F, C1E

Während meiner intensiven Auseinandersetzung mit Refactoring-Techniken habe ich ein umfassendes Verständnis für Strategien entwickelt, um Code lesbarer und verständlicher zu gestalten.

Extraktion von Funktionen/Methoden:

Die Aufteilung von großen Codeblöcken in kleinere Funktionen ermöglicht nicht nur eine klarere Struktur, sondern erleichtert auch das Verständnis.

```
# Vor Refactoring

def complex_task():

    # Langwieriger Codeblock


# Nach Refactoring

def helper_function_1():

    # Teilaufgabe 1


def helper_function_2():

    # Teilaufgabe 2


def complex_task():

    helper_function_1()

    helper_function_2()
```

Variablennamen verbessern:

Durch die Verwendung aussagekräftiger Variablennamen wird der Code nicht nur lesbarer, sondern auch leichter nachvollziehbar.

```
# Vor Refactoring

a = 10

b = 'Text'


# Nach Refactoring

user_age = 10

user_name = 'Text'
```

Magic Numbers ersetzen:

Die Ersetzung von magischen Zahlen durch benannte Konstanten verbessert die Lesbarkeit und erleichtert das Verständnis des Codes.

```
# Vor Refactoring

if value == 42:

  # ...


# Nach Refactoring

TARGET_VALUE = 42

if value == TARGET_VALUE:
```

Bedingte Anweisungen vereinfachen:

Die Vereinfachung von bedingten Anweisungen trägt dazu bei, den Code übersichtlicher zu gestalten und die Logik klarer zu verdeutlichen.

```
# Vor Refactoring

result = x if x > 0 else 0


# Nach Refactoring

result = max(x, 0)
```

Doppelte Codefragmente beseitigen:

Die Identifizierung und Beseitigung von Duplikaten verbessert nicht nur die Lesbarkeit, sondern fördert auch die Wartbarkeit des Codes.

```
# Vor Refactoring

def operation1():

   # Gemeinsamer Code


def operation2():

   # Gemeinsamer Code


# Nach Refactoring

def shared_operation():

   # Gemeinsamer Code


def operation1():

   shared_operation()


def operation2():

   shared_operation()
```

Codeverhalten einschätzen:

Vor einem Refactoring analysiere ich sorgfältig das bestehende Codeverhalten. Dies beinhaltet das Verständnis von Eingaben, Ausgaben und Seiteneffekten.

Testgetriebene Entwicklung (TDD):

Die Anwendung von TDD ermöglicht es mir, Tests vor dem Refactoring zu schreiben. Dadurch erhalte ich eine Sicherheitsnetz, um das gewünschte Verhalten sicherzustellen.

Regressionstests durchführen:

Nach jedem Refactoring führe ich Regressionstests durch, um sicherzustellen, dass bestehende Funktionalitäten nicht beeinträchtigt wurden.

Versionierung nutzen:

Die Verwendung von Versionskontrollsystemen ermöglicht es mir, Änderungen rückgängig zu machen, falls unerwartete Probleme auftreten.

Peer-Reviews und Feedback:

Ich integriere Peer-Reviews und Feedback in den Refactoring-Prozess, um verschiedene Perspektiven einzubeziehen und potenzielle Probleme frühzeitig zu erkennen.

Durch dieses tiefgreifende Verständnis kann ich nicht nur Code verbessern, sondern auch sicherstellen, dass Refactoring keine unerwünschten Nebeneffekte hat. Diese Herangehensweise gewährleistet, dass die Qualität des Codes bei jeder Veränderung erhalten bleibt oder sogar verbessert wird.

Diese Fähigkeiten haben meine Programmierpraxis auf ein höheres Niveau gehoben und werden auch in Zukunft eine zentrale Rolle in meinem Streben nach qualitativ hochwertigem Code spielen.