Lernnachweis zu Kompetenz C1G:

**Kompetenz**:

C1G:Ich kann einige Refactoring-Techniken aufzählen, die einen Code lesbarer und verständlicher machen.

**Lernnachweis**:

Während meines Studiums im Modul 323 habe ich verschiedene Refactoring-Techniken gelernt, die dazu dienen, den Code lesbarer und verständlicher zu machen. Hier sind einige dieser Techniken:

**Extrahieren von Funktionen/Methoden**:

**Ziel**: Reduzierung von Duplikation und Verbesserung der Lesbarkeit.

# Vor dem Refactoring

**def** **berechne\_gesamtpreis**(produkt, menge, preis):

**return** menge \* preis

**def** **zeige\_rechnung**(produkt, menge, preis):

gesamtpreis = berechne\_gesamtpreis(produkt, menge, preis)

**print**(f"Produkt: {produkt}, Menge: {menge}, Gesamtpreis: {gesamtpreis}")

# Nach dem Refactoring

**def** **berechne\_gesamtpreis**(menge, preis):

**return** menge \* preis

**def** **zeige\_rechnung**(produkt, menge, preis):

gesamtpreis = berechne\_gesamtpreis(menge, preis)

**print**(f"Produkt: {produkt}, Menge: {menge}, Gesamtpreis: {gesamtpreis}")

**Aufteilen von langen Funktionen/Methoden**:

**Ziel**: Verbesserung der Lesbarkeit und Wartbarkeit.

# Vor dem Refactoring

**def** **verarbeite\_bestellungen**(bestellungen):

**for** bestellung **in** bestellungen:

# Lange Logik hier...

# Nach dem Refactoring

**def** **verarbeite\_bestellungen**(bestellungen):

**for** bestellung **in** bestellungen:

verarbeite\_bestellung(bestellung)

**def** **verarbeite\_bestellung**(bestellung):

# Kürzere Logik hier...

**Verwendung von sprechenden Variablennamen**:

**Ziel**: Verbesserung der Verständlichkeit des Codes.

# Vor dem Refactoring

**def** **btr**(x, y):

**return** x + y

# Nach dem Refactoring

**def** **berechne\_summe**(zahl1, zahl2):

**return** zahl1 + zahl2

**Entfernen von unnötigem Code**:

**Ziel**: Vereinfachung des Codes und Verbesserung der Lesbarkeit.

# Vor dem Refactoring

**def** **verdopple\_zahl**(x):

**return** x \* **2** **if** x > **0** **else** x

# Nach dem Refactoring

**def** **verdopple\_zahl**(x):

**return** x \* **2**

**Für Flask Applikation**:

**http://localhost:5000/verarbeite\_daten** und **http://localhost:5000/verarbeite\_daten\_refactored**

**Wichtig**: Curl oder Postman und sende eine POST-Anfrage.

**Reflexion**:

Während meines Lernprozesses habe ich erkannt, dass Refactoring ein integraler Bestandteil des Entwicklungsprozesses ist. Die Fähigkeit, Code effektiv zu refaktorieren, führt zu einer verbesserten Wartbarkeit, Lesbarkeit und Verständlichkeit. Es ermöglicht auch die Anpassung des Codes an neue Anforderungen und Vermeidung von Redundanz.

**Fazit**:

Es ist wichtig, refaktorierten Code sorgfältig zu überprüfen und sicherzustellen, dass er weiterhin korrekt funktioniert. Tests können dabei eine entscheidende Rolle spielen, um die Funktionalität sicherzustellen.

Lernnachweis zu Kompetenz C1F:

**Kompetenz**:

C1F: Ich kann mit Refactoring-Techniken einen Code lesbarer und verständlicher machen.

**Lernnachweis**:

Während meines Studiums im Modul 323 habe ich gelernt, wie Refactoring-Techniken dazu beitragen können, den Code lesbarer und verständlicher zu machen. Hier präsentiere ich eine spezifische Technik in Form einer Beispielanwendung.

**Refactoring-Technik: Verwendung von List Comprehensions**

List Comprehensions sind eine elegante Möglichkeit, Listen in Python zu erstellen oder zu transformieren. Diese Technik führt zu kompakterem und lesbarerem Code.

**Vor dem Refactoring**:

# Funktion, die eine Liste der Quadratzahlen von 1 bis n erstellt

**def** **quadratzahlen\_bis\_n**(n):

quadratzahlen = []

**for** i **in** range(**1**, n+**1**):

quadratzahlen.append(i\*\***2**)

**return** quadratzahlen

**Nach dem Refactoring**:

# Funktion mit List Comprehension, um die Quadratzahlen von 1 bis n zu erstellen

**def** **quadratzahlen\_bis\_n**(n):

**return** [i\*\***2** **for** i **in** range(**1**, n+**1**)]

**Für Flask Applikation**:

**http://localhost:5000/quadratzahlen\_bis\_n/5** und **http://localhost:5000/quadratzahlen\_bis\_n\_refactored/5**

**Reflexion**:

Während meines Lernprozesses habe ich erkannt, dass die Anwendung von List Comprehensions und ähnlichen Techniken die Lesbarkeit des Codes erheblich verbessern kann. Solche Konstrukte ermöglichen es, komplexe Iterations- und Transformationslogik auf eine einzige Zeile zu reduzieren.

**Fazit**:

Es ist jedoch wichtig, ein Gleichgewicht zu finden und sicherzustellen, dass der Code auch für andere Entwickler verständlich bleibt. Manchmal kann zu viel Kompaktheit die Lesbarkeit beeinträchtigen, insbesondere wenn der Code zu komplex wird.

Lernnachweis zu Kompetenz C1E:

**Kompetenz**:

C1E: Ich kann die Auswirkungen des Refactorings auf das Verhalten des Codes einschätzen und sicherstellen, dass das Refactoring keine unerwünschten Nebeneffekte hat.

**Lernnachweis**:

Die Kompetenz C1E beinhaltet die Fähigkeit, die Auswirkungen von Refactoring auf das Verhalten des Codes zu beurteilen und sicherzustellen, dass das Refactoring keine unerwünschten Nebeneffekte hat. Dies erfordert eine gründliche Überprüfung des Codes nach dem Refactoring, um sicherzustellen, dass die Funktionalität intakt bleibt.

**Beispiel**:

**Angenommen, wir haben eine Funktion zur Berechnung der Quadratzahlen von 1 bis n, die ursprünglich so aussah**:

# Vor dem Refactoring

**def** **quadratzahlen\_bis\_n**(n):

quadratzahlen = []

**for** i **in** range(**1**, n+**1**):

quadratzahlen.append(i\*\***2**)

**return** quadratzahlen

**Nach dem Refactoring verwenden wir eine List Comprehension, um den Code zu vereinfachen**:

# Nach dem Refactoring

**def** **quadratzahlen\_bis\_n\_refactored**(n):

**return** [i\*\***2** **for** i **in** range(**1**, n+**1**)]

**Beurteilung und Sicherstellung**:

Um die Auswirkungen des Refactorings zu beurteilen und die Code-Stabilität sicherzustellen, können wir folgende Schritte unternehmen:

* Testen der Funktionalität
* Überprüfen auf unerwünschte Nebeneffekte
* Lesbarkeit und Wartbarkeit
* Code-Reviews
* Versionierung und Backups

Für Flask Applikation:

* http://localhost:5000/quadratzahlen\_bis\_n/5
* http://localhost:5000/quadratzahlen\_bis\_n\_refactored/5

**Reflexion**:

Die Fähigkeit, die Auswirkungen des Refactorings zu beurteilen und die Code-Stabilität sicherzustellen, ist entscheidend für einen reibungslosen Entwicklungsprozess. Durch sorgfältige Überprüfung und Tests können potenzielle Probleme frühzeitig erkannt und behoben werden.