AI Assisted Coding Ass-10.3

Code Review and Quality: Using AI to improve code quality and readability

2403A52L05 – B50

E. Sai Nithin Ramanujan

Task 1: AI-Assisted Bug Detection

Prompt and code:

```
#Task 1 - AI-Assisted Bug Detection (Lab 9)
"""
Prompt:generate a code to help checking this factorial code:
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
what you need to do is:
- test what it gives for factorial(5),
- tell me the exact bug and why it happens,
- give me the corrected code,
- and also check edge cases like 0, negative numbers, and non-integer input?

keep the explanation simple.
"""
def factorial_buggy(n: int) -> int:
    """Original buggy function from the assignment."""
    result = 1
    for i in range(1, n):
        result = result * i
    return result
def factorial_manual_fix(n: int) -> int:
    """Manual fix: include n in the loop range."""
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
def factorial_ai_fix(n: int) -> int:
    """AI-improved fix with validation and edge-case handling."""
    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    if n < 0:
        raise ValueError("factorial is not defined for negative numbers")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
```

```python
def run_task_1() -> None:
    print("TASK 1 - AI-Assisted Bug Detection")
    print("-" * 40)
    print("factorial_buggy(5):", factorial_buggy(5))
    print("factorial_manual_fix(5):", factorial_manual_fix(5))
    print("factorial_ai_fix(5):", factorial_ai_fix(5))

    print("\nBug identified:")
    print("Off-by-one error: range(1, n) excludes n, so multiplication misses the last value.")

    print("\nComparison:")
    print("Manual fix and AI fix both return 120 for n=5.")
    print("AI fix also handles edge cases (negative and non-integer inputs).")

    print("\nEdge-case checks:")
    print("factorial_ai_fix(0):", factorial_ai_fix(0))

    try:
        factorial_ai_fix(-3)
    except ValueError as exc:
        print("factorial_ai_fix(-3):", exc)

    try:
        factorial_ai_fix(3.5)
    except TypeError as exc:
        print("factorial_ai_fix(3.5):", exc)


if __name__ == "__main__":
    run_task_1()
```

Output:

```
PS D:\collage\AI-AC> python week-10/task-1.py
TASK 1 - AI-Assisted Bug Detection
----------------------------------------
factorial_buggy(5): 24
factorial_manual_fix(5): 120
factorial_ai_fix(5): 120

Bug identified:
Off-by-one error: range(1, n) excludes n, so multiplication misses the last value

Comparison:
Manual fix and AI fix both return 120 for n=5.
AI fix also handles edge cases (negative and non-integer inputs).

Edge-case checks:
factorial_ai_fix(0): 1
factorial_ai_fix(-3): factorial is not defined for negative numbers
factorial_ai_fix(3.5): n must be an integer
```

Task 2: Improving Readability & Documentation

Prompt and code:

```
#Task 2 - Improving Readability & Documentation (Lab 9)
"""
Prompt: generate a code to improve this function readability and documentation:
def calc(a, b, c):
    if c == "add":
        return a + b
    elif c == "sub":
        return a - b
    elif c == "mul":
        return a * b
    elif c == "div":
        return a / b
what you need to do is:
- critique readability, naming, and lack of documentation,
- rewrite with better names,
- add full docstring (description, params, return, examples),
- add division by zero handling,
- add input validation,
- test valid and invalid inputs.

keep the explanation simple.
"""
def calc(a, b, c):
    """Original poorly written function from the assignment."""
    if c == "add":
        return a + b
    elif c == "sub":
        return a - b
    elif c == "mul":
        return a * b
    elif c == "div":
        return a / b
    return None

def calculate_numbers(first_number, second_number, operation):
    """Perform a basic arithmetic operation.

    Parameters:
        first_number (int or float): First numeric value.
        second_number (int or float): Second numeric value.
        operation (str): One of "add", "sub", "mul", "div".

    Returns:
        int or float: Computed result.

    Raises:
        TypeError: If inputs are invalid types.
        ValueError: If operation is unsupported.
        ZeroDivisionError: If division by zero is attempted.

    Examples:
        >>> calculate_numbers(10, 5, "add")
        15
        >>> calculate_numbers(10, 5, "sub")
        5
        >>> calculate_numbers(10, 5, "mul")
        50
        >>> calculate_numbers(10, 5, "div")
        2.0
    """
    if not isinstance(operation, str):
        raise TypeError("operation must be a string")

    if not isinstance(first_number, (int, float)) or isinstance(first_number, bool):
        raise TypeError("first_number must be an int or float")

    if not isinstance(second_number, (int, float)) or isinstance(second_number, bool):
        raise TypeError("second_number must be an int or float")

    op = operation.strip().lower()
```

```python
    if op == "add":
        return first_number + second_number
    if op == "sub":
        return first_number - second_number
    if op == "mul":
        return first_number * second_number
    if op == "div":
        if second_number == 0:
            raise ZeroDivisionError("Cannot divide by zero")
        return first_number / second_number

    raise ValueError("operation must be one of: add, sub, mul, div")
def run_task_2() -> None:
    print("TASK 2 - Improving Readability & Documentation")
    print("-" * 40)
    print("Original calc(10, 2, 'div'):", calc(10, 2, "div"))
    print("Improved calculate_numbers(10, 2, 'div'):", calculate_numbers(10, 2, "div"))
    print("\nCritique:")
    print("Original function uses unclear names (a, b, c), has no docstring, and weak error handling.")
    print("\nComparison:")
    print("Improved version uses descriptive names, docstring, validation, and explicit exceptions.")
    print("\nInvalid input checks:")
    try:
        calculate_numbers(5, 0, "div")
    except ZeroDivisionError as exc:
        print("calculate_numbers(5, 0, 'div'):", exc)
    try:
        calculate_numbers(5, 2, 123)
    except TypeError as exc:
        print("calculate_numbers(5, 2, 123):", exc)
    try:
        calculate_numbers("5", 2, "add")
    except TypeError as exc:
        print("calculate_numbers('5', 2, 'add'):", exc)

if __name__ == "__main__":
    run_task_2()
```

Output:

```
PS D:\collage\AI-AC> python week-10/task-2.py
TASK 2 - Improving Readability & Documentation
----------------------------------------
Original calc(10, 2, 'div'): 5.0
Improved calculate_numbers(10, 2, 'div'): 5.0

Critique:
Original function uses unclear names (a, b, c), has no docstring, and weak error handling.

Comparison:
Improved version uses descriptive names, docstring, validation, and explicit exceptions.

Invalid input checks:
calculate_numbers(5, 0, 'div'): Cannot divide by zero
calculate_numbers(5, 2, 123): operation must be a string
calculate_numbers('5', 2, 'add'): first_number must be an int or float
```

Task 3: Enforcing Coding Standards

```python
#Task 3 - Enforcing Coding Standards (Lab 9)
"""Prompt: review prime-check code for PEP8 issues, refactor it, and verify behavior."""
def Checkprime(n):
    """Original non-PEP8 code from the assignment."""
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def check_prime(number):
    """Return True if number is prime, else False."""
    if not isinstance(number, int):
        raise TypeError("number must be an integer")
    if number < 2:
        return False
    for divisor in range(2, int(number ** 0.5) + 1):
        if number % divisor == 0:
            return False
    return True
def run_task_3() -> None:
    print("TASK 3 - Enforcing Coding Standards")
    print("-" * 40)
    print("Original Checkprime(11):", Checkprime(11))
    print("Refactored check_prime(11):", check_prime(11))
    print("\nPEP8 issues:")
    print("- Function name should be lowercase_with_underscores (Checkprime -> check_prime)")
    print("- Missing type validation in original function")
    print("- Original logic does not handle n < 2 correctly")
    print("\nFunctionality check:")
    print("check_prime(2):", check_prime(2))
    print("check_prime(4):", check_prime(4))
    print("check_prime(17):", check_prime(17))
    print("check_prime(1):", check_prime(1))
    print("check_prime(0):", check_prime(0))
    print("check_prime(-5):", check_prime(-5))
    print("\nTeam note:")
    print("AI-assisted reviews can quickly flag style issues and basic logic bugs before human review.")

if __name__ == "__main__":
    run_task_3()
```

Output:

```
PS D:\collage\AI-AC> python week-10/task-3.py
TASK 3 - Enforcing Coding Standards
----------------------------------------
Original Checkprime(11): True
Refactored check_prime(11): True

PEP8 issues:
- Function name should be lowercase_with_underscores (Checkprime -> check_prime)
- Missing type validation in original function
- Original logic does not handle n < 2 correctly

Functionality check:
check_prime(2): True
check_prime(4): False
check_prime(17): True
check_prime(1): False
check_prime(0): False
check_prime(-5): False

Team note:
AI-assisted reviews can quickly flag style issues and basic logic bugs before human review.
```

Task 4: AI as a Code Reviewer in Real Projects

Prompt and code:

```python
#Task 4 - AI as a Code Reviewer in Real Projects (Lab 9)
"""Prompt: review processData, improve naming/validation, and make multiplier configurable."""
def processData(d):
    return [x * 2 for x in d if x % 2 == 0]
def multiply_even_numbers(numbers, multiplier=2):
    if not isinstance(numbers, list):
        raise TypeError("numbers must be a list")
    if not isinstance(multiplier, (int, float)) or isinstance(multiplier, bool):
        raise TypeError("multiplier must be an int or float")
    result = []
    for value in numbers:
        if not isinstance(value, (int, float)) or isinstance(value, bool):
            raise TypeError("all items in numbers must be int or float")
        if value % 2 == 0:
            result.append(value * multiplier)
    return result
def run_task_4() -> None:
    print("TASK 4 - AI as a Code Reviewer")
    print("-" * 40)
    print("Original processData([1, 2, 3, 4]):", processData([1, 2, 3, 4]))
    print("Improved multiply_even_numbers([1, 2, 3, 4], 3):", multiply_even_numbers([1, 2, 3, 4], 3))
    print("\nManual review:")
    print("- Name processData is unclear")
    print("- Hardcoded multiplier reduces reusability")
    print("- No input validation for bad data")
    print("\nEdge-case checks:")
    print("multiply_even_numbers([]):", multiply_even_numbers([]))
    try:
        multiply_even_numbers("not_a_list")
    except TypeError as exc:
        print("multiply_even_numbers('not_a_list'):", exc)
    try:
        multiply_even_numbers([2, "x", 4])
    except TypeError as exc:
        print("multiply_even_numbers([2, 'x', 4]):", exc)
    print("\nReflection:")
    print("AI should assist reviewers, but final review should stay with humans for context and design decisions.")
if __name__ == "__main__":
    run_task_4()
```

Output:

```
PS D:\collage\AI-AC> python week-10/task-4.py
TASK 4 - AI as a Code Reviewer
----------------------------------------
Original processData([1, 2, 3, 4]): [4, 8]
Improved multiply_even_numbers([1, 2, 3, 4], 3): [6, 12]

Manual review:
- Name processData is unclear
- Hardcoded multiplier reduces reusability
- No input validation for bad data

Edge-case checks:
multiply_even_numbers([]): []
multiply_even_numbers('not_a_list'): numbers must be a list
multiply_even_numbers([2, 'x', 4]): all items in numbers must be int or float

Reflection:
AI should assist reviewers, but final review should stay with humans for context and design decisions.
```

Task 5: AI-Assisted Performance Optimization

Prompt and code:

```python
#Task 5 - AI-Assisted Performance Optimization (Lab 9)
"""Prompt: analyze sum_of_squares complexity, optimize it, and compare runtime."""
from time import perf_counter

def sum_of_squares(numbers):
    """Original version."""
    total = 0
    for num in numbers:
        total += num ** 2
    return total
def sum_of_squares_optimized(numbers):
    """Short optimized version."""
    return sum(x * x for x in numbers)
def benchmark(size=1_000_000, repeats=3):
    """Return average runtime of original and optimized functions."""
    data = list(range(size))
    t1 = 0.0
    t2 = 0.0
    result_1 = 0
    result_2 = 0
    for _ in range(repeats):
        start = perf_counter()
        result_1 = sum_of_squares(data)
        t1 += perf_counter() - start

        start = perf_counter()
        result_2 = sum_of_squares_optimized(data)
        t2 += perf_counter() - start
    return t1 / repeats, t2 / repeats, result_1, result_2
```

```python
def run_task_5() -> None:
    print("TASK 5 - AI-Assisted Performance Optimization")
    print("-" * 40)
    print("Time complexity:")
    print("- Original: O(n)")
    print("- Optimized: O(n)")
    original_time, optimized_time, original_result, optimized_result = benchmark()
    print("\nResult check:", original_result == optimized_result)
    print(f"Original time:  {original_time:.6f} seconds")
    print(f"Optimized time: {optimized_time:.6f} seconds")
    speedup = original_time / optimized_time if optimized_time > 0 else float("inf")
    print(f"Speedup factor: {speedup:.2f}x")
    print("\nTrade-off:")
    print("Optimized code is shorter and readable; speed can vary by environment.")

if __name__ == "__main__":
    run_task_5()
```

Output:

```
PS D:\collage\AI-AC> python week-10/task-5.py
TASK 5 - AI-Assisted Performance Optimization
----------------------------------------
Time complexity:
- Original: O(n)
- Optimized: O(n)

Result check: True
Original time:  0.063500 seconds
Optimized time: 0.063998 seconds
Speedup factor: 0.99x

Trade-off:
Optimized code is shorter and readable; speed can vary by environment.
```