Code Analysis Report

Analyzed Code:

```
from utility import *
from Employee import *

class EmployeesManager:
def __init__(self):
self.employees = []

def add_employee(self):
print('Enter employee data')
name = input("Enter employee name")
age = input_is_valid("Enter employee age: ")
salary = input_is_valid("Enter employee salary: ")
self.employees.append(Employee(name, age, salary))

def list_employee(self):
if len(self.employees) == 0:
print("\nEmployee list is empty !")
return
else:
for emp in self.employees:
print(emp)

def delete_employees_with_age(self, age_from, age_to):
for emp in self.employees:
if age_from <= emp.age <= age_to:
print(f"\tDeleting employee {emp.name}")
self.employees.remove(emp)

def find_employee_by_name(self, name):
for emp in self.employees:
if emp.name == name:
return emp
return None

def update_salary_by_name(self, name, salary):
emp = self.find_employee_by_name(name)
```

```
if emp is None:
print('Error: No employee found')
else:
emp.salary = salary
```

Static Code Metrics:

Metric	Value
Lines of Code (LOC)	34
Comment Density	0.0
Cyclomatic Complexity	16
Maintainability Index	52.91001111636902
Halstead Metrics	
Unique Operators	3
Unique Operands	8
Total Operators	5
Total Operands	9
Volume	48.43204266092217
Effort	81.72907199030617
Depth of Inheritance Tree (DIT)	0
Coupling Between Object classes (CBO)	11
Lack of Cohesion of Methods (LCOM)	0.0
Fan-in	10
Fan-out	10
Number of Methods (NOM)	6

McCall's Intermediate Quality Metrics:

Metric	Value
Modifiability	0.43
Testability	0.6
Reliability	0.41
Understandability	0.54
Self-Descriptiveness	0.02

Reusability	0.72
Portability	0.88
Efficiency	0.83

GPT Analysis Report:

The provided Python program encompasses an `EmployeesManager` class designed for managing employees. Below is a comprehensive analysis of the code along with specific suggestions for enhancements in terms of structure, readability, and performance.

1. Comment Density and Documentation

Current Status: The Comment Density is 0.0, and there are no comments or docstrings explaining the purpose of classes or methods.

Recommendations:

- Include docstrings for each class and method. Documentation should explain the purpose, parameters, return values, and exceptions raised. This will improve understandability and aid future developers in grasping the code's intent quickly.

- Example for `add_employee`:

```
python
def add_employee(self):
"""Prompts the user to enter the employee's name, age, and salary, and adds the employee to the list."""
```

2. Input Validation

Current Status: The use of `input_is_valid` is good, but without context provided in the given code, it is unclear how inputs are validated (e.g., checking if age and salary are numeric).

Recommendations:

- Ensure that input validation is robust and clearly defined. If `input_is_valid` isn't handling exceptions or invalid cases effectively, consider improving it.

- Example input validation could also give feedback on invalid inputs and keep the user in a loop until valid input is entered.

```
python
def input_is_valid(prompt):
while True:
try:
value = float(input(prompt))
return value
```

```
except ValueError:
print("Invalid input. Please enter a numeric value.")
```

3. Method Complexity

Current Status: The cyclomatic complexity is quite high (16), indicating that methods have multiple paths and decisions.

Recommendations:

- Simplify methods, especially `delete_employees_with_age`. This method can be rewritten using list comprehension to make it more efficient and readable.

```
python
def delete_employees_with_age(self, age_from, age_to):
self.employees = [emp for emp in self.employees if not (age_from <= emp.age
<= age_to)]
```

- Consider separating logic into smaller methods, especially if certain blocks of code can be reused elsewhere.

4. Error Handling

Current Status: The code contains basic error handling, but it could be improved significantly.

Recommendations:

- Raise custom exceptions or at least define consistent error messages to aid debugging and provide clearer information to users.

- Implement a logging mechanism to capture errors instead of solely relying on print statements, especially for exception reporting.

5. List Manipulation

Current Status: The use of `self.employees.remove(emp)` could lead to problems when looping through a list, as modifying a list while iterating can skip elements or cause runtime errors.

Recommendations:

- Instead of removing employees while iterating through them, construct a new list excluding the employees you want to delete. This can prevent unintended behavior:

```
python
def delete_employees_with_age(self, age_from, age_to):
self.employees = [emp for emp in self.employees if not (age_from <= emp.age
<= age_to)]
```

6. Employee Class Design

Current Status: The `Employee` class is referenced but not defined in the provided code. Its structure is critical for further improvements.

Recommendations:

- Consider using properties (`@property`) for encapsulating attributes like `age` and `salary` in the `Employee` class. This can add input validation and maintain class invariants.

- Ensure that Employee data is immutable where necessary, or provide methods to update employees rather than directly modifying attributes.

7. Code Organization and Interface

Current Status: Utility imports and Employee import are not clear in terms of organization.

Recommendations:

- Clearly define what's contained in `utility` and `Employee` modules. Minimize the use of wildcard imports (`from module import *`) to prevent polluting the namespace and reduce ambiguity about where functions are coming from.

- Use explicit imports such as `from utility import input_is_valid`.

Summary of Metrics

While the code has a reasonable foundation, the metrics demonstrate areas needing improvement, particularly in maintainability, understandability, and reliability. The suggestions above aim to reduce cyclomatic complexity, improve comment density, enhance error handling, and promote good software practices.

By implementing the improvements outlined, the overall quality and maintainability of the code would greatly enhance, thus resulting in a more robust and user-friendly EmployeesManager class.