

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника

(наименование факультета)

Кафедра Кибербезопасность информационных систем

(наименование кафедры)

**Практическое занятие № 3**

на тему «Оценка характеристик программ   
на основе объектно-ориентированных метрик Мартина»

Выполнил обучающийся гр. ВКБ42

Ковалев Данил Петрович

Проверил:

доцент кафедры «КБИС» Куликова Ольга Витальевна

Ростов-на-Дону

2025

**Цель работы:**

Изучить методику оценки характеристик программ с использованием объектно-ориентированных метрик Мартина.

**Контрольные вопросы:**

1. В чем заключается суть исследования программы при объектно-ориентированном программировании?

В программах класс редко может быть повторно использован изолированно от других классов. Почти всегда класс имеет группу классов, с которыми он работает во взаимодействии и от которых его достаточно трудно отделить. Для повторного использования таких классов необходимо заново использовать всю группу классов. Связность такой группы классов (она называется категорией классов) достаточно высока, и для ее существования должны быть соблюдены некоторые условия.

1. Какие условия должны быть соблюдены для существования категории классов?

• *классы в пределах категории закрыты от любых попыток изменения отдельных экземпляров.* Это означает, что если одному классу необходимо измениться, то весьма вероятно изменение всех классов в этой категории. Если любой из классов открыт для некоторого вида изменений, они все открыты для этого вида изменений;

• *классы в категории повторно используются только вместе.* Они настолько взаимозависимы, что не могут быть отделены друг от друга. Поэтому если предпринимается попытка повторного использования одного класса в категории, то все другие классы этой категории также повторно используются вместе с таким классом;

• *классы в категории обеспечивают некоторую общую функцию или достигают некоторую общую цель*.

1. Какие метрики используются для оценки характеристик программы при объектно-ориентированном программировании?

• **(центростремительное сцепление)** - метрика, определяющая количество классов вне конкретной категории, которые зависят от классов внутри её;

• **(центробежное сцепление)** - метрика, оценивающая количество классов внутри конкретной категории, которые зависят от классов вне её;

• ***I* (нестабильность)** - расчетная метрика, определяемая в соответствии с выражением

Данная метрика имеет диапазон значений [0, 1]: значение метрики  
означает максимально стабильную категорию, а указывает максимально нестабильную категорию.

1. Что такое «мера абстрактности» в метриках Мартина?

Дополнительно к указанным метрикам можно определять меру ***А*** (абстрактность), которая позволяет оценить абстрактность категории (если категория абстрактна, то она является достаточно гибкой и может быть легко расширена):

где - количество абстрактных классов в категории; - общее количество классов в категории.

Значения метрики абстрактности расположены в диапазоне [0, 1]: при нулевом значении *категория полностью конкретна*, а при единичном значении является *полностью абстрактной*.

1. Что такое главная последовательность программы в метриках Мартина?

На основе приведенных метрик Мартина можно построить график, отражающий зависимость между абстрактностью и нестабильностью. Если построить прямую, задаваемую формулой ***I + А = 1***, то категории, расположенные на этой прямой, будут иметь наилучшую сбалансированность между абстрактностью и нестабильностью. Эта прямая называется **главной последовательностью**.

Получив главную последовательность, можно ввести еще две метрики:

• расстояние до главной последовательности:

• нормализированное расстояние до главной последовательности:

Практически для любых категорий классов справедливым является следующее утверждение: чем ближе они находятся к главной последовательности, тем лучше для обеспечения качества, и как следствие надежности, программного средства.

**Задания:**

При оценке характеристик программ на основе объектно-ориентированных метрик Мартина необходимо выполнить следующее:

* 1. разработать программу, реализующую заданный, в соответствии с вариантом, алгоритм (рекомендуется использовать язык программирования С#);
  2. оценить характеристики разработанной программы на основе применения объектно-ориентированных метрик Мартина.

Практическая работа выполняется в соответствии с вариантом (номер варианта соответствует номеру студента в журнале группы).

**Вариант 2:**

Определить понятие «Радиостанция». Состояние объекта определяется следующими полями:

* наименование радиостанции (строка до 60 символов);
* частота вещания (длинное целое число).

Наименование радиостанции может иметь несколько слов, разделенных пробелами. В таблице радиостанций изменить название радиостанции, вещающей на заданной частоте.

**Реализация программы**

|  |  |
| --- | --- |
| **Номера строк** | **Строки программы** |
| 1 | from abc import abstractmethod |
| 2 | from collections import defaultdict |
| 3 | from typing import Protocol |
| 4 |  |
| 5 | class RadioStation(Protocol): |
| 6 | @property |
| 7 | @abstractmethod |
| 8 | def name(self) -> str: |
| 9 | raise NotImplementedError() |
| 10 |  |
| 11 | @name.setter |
| 12 | @abstractmethod |
| 13 | def name(self, name: str) -> None:  raise NotImplementedError() |
| 14 |  |
| 15 | @property |
| 16 | @abstractmethod |
| 17 | def frequency(self) -> int: |
| 18 | raise NotImplementedError() |
| 19 |  |
| 20 | @frequency.setter |
| 21 | @abstractmethod |
| 22 | def frequency(self, frequency: int) -> None: |
| 23 | raise NotImplementedError() |
| 24 |  |
| 25 | class SimpleRadioStation(RadioStation): |
| 26 | def \_\_init\_\_(self, name: str, frequency: int) -> None: |
| 27 | self.name: str = name |
| 28 | self.frequency: int = frequency |
| 29 |  |
| 30 | @property |
| 31 | @override |
| 32 | def frequency(self) -> int:  return self.\_\_frequency |
| 33 |  |
| 34 | @frequency.setter |
| 35 | @override |
| 36 | def frequency(self, frequency: int) -> None: |
| 37 | self.\_\_frequency: int = frequency |
| 38 |  |
| 39 | @property |
| 40 | @override |
| 41 | def name(self) -> str: |
| 42 | return self.\_\_name |
| 43 |  |
| 44 | @name.setter |
| 45 | @override |
| 46 | def name(self, new\_name: str) -> None: |
| 47 | if len(new\_name) >= 60: |
| 48 | Raise ValueError(“Длина имени не должна превышать 60 символов”) |
| 49 | self.\_\_name: str = new\_name |
| 50 |  |
| 51 | class RadioStationManager: |
| 52 | def \_\_init\_\_(self) -> None: |
| 53 | self.\_stations: List[RadioStation] = [] |
| 54 |  |
| 55 | def add\_station(self, station: RadioStation) -> None: |
| 56 | self.\_stations.append(station) |
| 57 |  |
| 58 | def update\_frequency\_by\_name(self, target\_name: int, new\_frequency: int) -> int: |
| 59 | count: int = 0 |
| 60 | for station in self.\_stations: |
| 61 | if station.name == target\_name: |
| 62 | station.frequency = new\_frequency; count += 1 |
| 63 | return count |
| 64 |  |
| 65 | @property |
| 66 | def stations(self) -> Iterable[RadioStation]: |
| 67 | return self.\_stations |
| 68 |  |
| 69 |  |
| 70 |  |
| 71 |  |
| 72 | def main() -> None: |
| 73 | manager: RadioStationManager = RadioStationManager() |
| 74 | try: |
| 75 | n: int = int(input(“Введите количество радиостанций: ”) |
| 76 |  |
| 77 | for i in range(n): |
| 78 | name: str = input(f”Радиостанция {i+1}. Наименование: ”) |
| 79 | while True: |
| 80 | try: |
| 81 | freq: int = int(input(“Частота (целое число): ”)) |
| 82 | break |
| 83 | except ValueError: |
| 84 | print(“Введите целое число”) |
| 85 | manager.add\_station(SimpleRadioStation(name, freq)) |
| 86 | print(“\n Введите частоту для изменения названия”) |
| 87 | target\_freq: int = int(input(“Целевая частота: ”)) |
| 88 | new\_name: str = input(“Новое название: ”) |
| 89 |  |
| 90 | updated\_count: int = manager.update\_name\_by\_frequency(target\_freq, new\_name) |
| 91 | print(f”\nОбновлено {updated\_count} радиостанций”) |
| 92 |  |
| 93 | print(“\nТекущий список радиостанций:”) |
| 94 | for station in manager.stations: |
| 95 | print(f“{station.name} ({station.frequency} кГц)”) |
| 96 | except ValueError as e: |
| 97 | print(f“Ошибка ввода: {e}”) |
| 98 |  |
| 99 | if \_\_name\_\_ == “\_\_main\_\_”: |
| 100 | main() |
|  |  |

**Оценка характеристик программы**

Проанализируем текст программы для оценки ее качества с помощью метрик Мартина, которые позволяют оценить меру сложности объектно-ориентированной программы на основе анализа организационной структуры классов программы и их связности, так как все классы работают во взаимодействии.

Исходный код программы включает три класса, которые можно отнести к одной категории, предназначенной для решения задачи поиска радиостанций в заданном диапазоне частот:

К ним относятся следующие классы:

***class RadioStation*** (строка 5) – абстрактный протокол, определяющий интерфейс для проверки попадания частоты в диапазон;

***class SimpleRadioStation*** (строка 25) – конкретная реализация радиостанции с полями name и frequency;

***class RadioStationManager*** (строка 51) ***–*** класс для управления коллекцией радиостанций и изменения названий по частоте.

***def main*** (строка 72) – реализует взаимодействие с пользователем, но не входит в категорию, так как не содержит бизнес-логики.

**Центростремительное сцепление (*Ca*)**

Классы других категорий (например, стандартные библиотеки Python) не зависят от классов данной категории. ***Ca = 0*** — полная независимость от внешних категорий.

**Центробежное сцепление (*Ce*)**

Классы категории зависят от внешних модулей:

1. ***RadioStation*** использует ***Protocol*** и ***abstractmethod, property*** из стандартной библиотеки.
2. ***SimpleRadioStation*** использует ***property, override*** из стандартной библиотеки.
3. ***RadioStationManager*** использует ***Mapping, List*** из модуля ***typing,*** а также ***property*** из стандартной библиотеки.
4. Функция ***main()*** использует ***input()***, ***print()***, ***float()*** и обработку исключений. Так как все три класса категории зависят от внешних модулей, ***Ce = 3***.

***Нестабильность (I)***

Центробежное сцепление классов по теории Мартина **Се = *3***. Расчетная метрика нестабильности **I** определяется следующим образом:

Исходя из полученного значения (***I = 1***) следует, что категория классов анализируемой программы является максимально нестабильной, так как зависимость от классов других категорий носит преобладающий характер.

***Абстрактность (A)***

Определим меру абстрактности **А** рассматриваемой категории классов. В данной имплементации у нас есть один абстрактный протокол – ***RadioStation***, следовательно n**А *= 1***. Общее количество классов в рассматриваемой категории составляет 3, следовательно, ***nAll = 3***.

Исходя из значения метрики **А *= 0.333*** можно сделать вывод, что рассматриваемая категория частично абстрактна. Это позволяет расширить функционал через реализацию интерфейса ***RadioStation***.

***Расстояние до главное последовательности (D)***

Расстояние до главной последовательности определяется уравнением   
***I +* А = *1***. В соответствии с теорией Мартина расстояние до главной последовательности определятся формулой:

Категория расположена на прямой главной последовательности, что свидетельствует о наилучшей сбалансированности между абстрактностью и нестабильностью. Нормализованное расстояние до главной последовательности равно:

**Вывод:** Из полученных результатов анализа текста программы следует, что в целом качество разработанной, программы можно считать достаточно высоким, так как рассматриваемая категория классов исходного кода находится в пределах нормализованного расстояния до главной последовательности.