

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 «Книга». Состояние объекта определяется следующими полями:

* единый регистрационный номер;
* автор (строка до 20 симовлов);
* год издания (целое число);
* количество экземпляров (целое число);

Вычислить количество экземпляров книг заданного автора, выпущенных в период с 2007г. по 2016г.

**Реализация программы**

|  |  |
| --- | --- |
| **Номера строк** | **Строки программы** |
| 1 | from abc import abstractmethod |
| 2 | from typing import Protocol |
| 3 |  |
| 4 |  |
| 5 | class Book(Protocol): |
| 6 | @property |
| 7 | @abstractmethod |
| 8 | def reg\_number(self) -> int: |
| 9 | raise NotImplementedError() |
| 10 |  |
| 11 | @reg\_number.setter |
| 12 | @abstractmethod |
| 13 | def reg\_number(self, value: int) -> None:  raise NotImplementedError() |
| 14 |  |
| 15 | @property |
| 16 | @abstractmethod |
| 17 | def author(self) -> int: |
| 18 | raise NotImplementedError() |
| 19 |  |
| 20 | @author.setter |
| 21 | @abstractmethod |
| 22 | def author(self, value: str) -> None: |
| 23 | raise NotImplementedError() |
| 24 |  |
| 25 | @property |
| 26 | @abstractmethod |
| 27 | def year(self) -> int |
| 28 | raise NotImplementedError() |
| 29 |  |
| 30 | @property |
| 31 | @abstractmethod |
| 32 | def copies(self) -> int:  raise NotImplementedError() |
| 33 |  |
| 34 | @copies.setter |
| 35 | @abstractmethod |
| 36 | def copies(self, value: int) -> None: |
| 37 | raise NotImplementedError() |
| 38 |  |
| 39 | class SimpleBook(Book): |
| 40 | def \_\_init\_\_(self, reg\_number: int, author: str, year: int, copies: int) -> None: |
| 41 | self.reg\_number: int = reg\_number |
| 42 | self.author: str = author |
| 43 | self.year: int = year |
| 44 | self.copies: int = copies |
| 45 |  |
| 46 | @property |
| 47 | @override |
| 48 | def reg\_number(self) -> int: |
| 49 | return self.\_reg\_number |
| 50 |  |
| 51 | @reg\_number.setter |
| 52 | @override |
| 53 | def reg\_number(self, value: int) -> None: |
| 54 | if value <= 0: |
| 55 | raise ValueError(“Регистрационный номер должен быть положительным”) |
| 56 | self.\_reg\_number: int = value |
| 57 |  |
| 58 | @property |
| 59 | @override |
| 60 | def author(self, value: str) -> None: |
| 61 | if len(value) >= 20: |
| 62 | raise ValueError(“Имя автора не должно превышать 20 символов”) |
| 63 |  |
| 64 | @property |
| 65 | @override |
| 66 | def year(self) -> int: |
| 67 | return self.\_year |
| 68 |  |
| 69 | @property |
| 70 | @overide |
| 71 | def copies(self) -> int: |
| 72 | return self.\_copies |
| 73 |  |
| 74 | @copies.setter |
| 75 | @override |
| 76 | def copies(self, value: int) -> None: |
| 77 | if value < 0: |
| 78 | raise ValueError(“Количество экземпляров не может быть отрицательным”)  self.\_copies: int = value |
| 79 |  |
| 80 | class BookManager: |
| 85 | def \_\_init\_\_(self) -> None: |
| 86 | self.\_books: List[Book] = [] |
| 87 |  |
| 88 | def add\_book(self, book: Book) -> None: |
| 89 | self.\_books.append(book) |
| 90 |  |
| 91 | def count\_books\_by\_author(self, target\_author: str) -> int: |
| 92 | count: int = 0 |
| 93 | for book in self.\_books: |
| 94 | if book.author == target\_author and 2007 <= book.year <= 2016: |
| 95 | count += book.copies |
| 96 | return count |
| 97 |  |
| 98 | def main() -> None: |
| 99 | manager: BookManager = BookManager() |
| 100 | try: |
| 101 | n: int = int(input(“Введите количество книг: ”)) |
| 102 |  |
| 103 | for i in range(n): |
| 104 | while True: |
| 105 | try: |
| 106 | reg\_number: int = int(input(“Книга {i+1}. Регистрационный номер: ”)) |
| 107 | author: str = input(“Автор: ”) |
| 108 | year: int = int(input(“Год издания: “)) |
| 109 | copies: int = int(input(“Количество экземпляров: ”)) |
| 110 | book: SimpleBook = SimpleBook(reg\_number, author, year, copies) |
| 111 | break |
| 112 | manager.add\_book(book) |
| 113 | except ValueError as e:  print(f“Ошибка ввода: {e}. Повторите ввод!”) |
|  |  |
| 115 |  |
|  |  |
| 116 | if \_\_name\_\_ == “\_\_main\_\_”: |
| 117 | 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 Book*** (строка 5) – абстрактный протокол, определяющий интерфейс для работы с книгами;

***class SimpleBook*** (строка 39) – конкретная реализация книги с валидацией данных;

***class BookManager*** (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. ***RadioSt*** 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.