ПЗ 4. Объектно-ориентированные метрики: метрики Мартина.

(Составили Никонова Г.В. ngvlad@mail.ru, Никонов А.В. nalva@mail.ru)

ЭЛЕКТРОННЫЙ ВАРИАНТ

https://disk.yandex.ru/d/xgBQ5OhfmR1fdg

Тематика: оценка качества продуктов объектно-ориентированного программирования (ООП). Метрики Мартина.

ЗАДАНИЕ

1. Выбрать программу, ОБЯЗАТЕЛЬНО реализованную в парадигме ООП на знакомом языке высокого уровня. Разобрать её работу, и представить схему алгоритма программы.

Парадигма ООП: при разработке **программы, она рассматривается как совокупность взаимодействующих между собой объектов**. (**Парадигма** – это **модель, образец или система принципов (взглядов), на которых строится деятельность в какой-либо области**). **Эти объекты обладают свойствами (атрибутами)** и **методами (действиями (функциями), и организуются в классы (шаблоны)**. ООП делает код более модульным и понятным.

Класс, как шаблон или чертёж для создания объектов, **описывает их структуру и поведение**. *Например*, класс «**Лиса**» может описывать **свойства** в виде **цвета** животного и **его породы**, а также **методы** – как «**накормить**», как «**напоить**», как «**вычесать**», и т. п.

Объект – это **экземпляр, являющийся конкретным представителем класса**. *Напр.*, «**Чёрно-бурая лиса в зоопарке**»: это **объект, созданный по шаблону класса «Лиса»**.

Свойства (**атрибуты**) объекта: это **данные, хранящиеся в объекте, и они определяют его состояние**. В примере с объектом «**Чёрно-бурая лиса**», это могут быть **цвет**, **дата рождения**, **окрас** животного.

Методы (**действия** (**функции**): это **функции, которые могут быть вызваны у объекта и способны изменять его** (объекта) **состояние**, или **выполнять действия**. *Напр.*, метод «**накормить**» изменяет состояние лисицы.

Преимущества ООП заключаются **в «Модульности»**, когда **программа разбивается на отдельные независимые объекты, что упрощает её структуру**.

Представить графически схему алгоритма программы [1]. **Установочный материал см. в приложении А. Листинг программы при-**

вести в приложении.

2. Представить графически и кратко описать «диаграмму классов программы» в нотации языка UML (Unified Modeling Language – унифицированный язык моделирования). Это не язык программирования.

В диаграмме классов программы **показываются собственно объекты программы и классы, связи между ними, зависимости и атрибуты**.

Построить UML-диаграмму **можно двумя средствами (рекомендуется первое средство): 1) с помощью любого сервиса для построения диаграмм – например бесплатный онлайн сервис Diagrams.net (Draw.io) (https://app.diagrams.net/). Дают возможность рисовать диаграммы вручную: пользователь проектирует диаграмму, подписывает элементы, оставляет заметки.** Также возможен сервис **бесплатный для некоммерческого использования: Visual Paradigm Online (https://online.visual-paradigm.com/ru/) (возможно понадобится vpn);**

2) с помощью программных модулей: модули для интегрированных сред разработки (IDE) и библиотеки для ряда языков программирования **позволяют создавать UML-диаграммы с помощью программного кода.** Здесь также **есть возможность сгенерировать диаграмму на основе имеющегося кода программы.**

Пример **построения «диаграммы классов» некой абстрактной программы** приведён **в приложении Б**.

3. Рассчитать пять метрик Мартина для выбранной программы. Программа индивидуальна для каждого студента. Результаты расчётов внести в итоговую сводную таблицу. *Пример* **расчёта метрик Мартина для некой программы** приведён **в приложении В.**

4. На основе метрик Мартина построить рисунок, на котором **показана линия** как геометрическое место точек, **являющихся суммой абстрактности и нестабильности** компонентов программы. На рисунке также **показать линию**, заданную суммой **абстрактности и нестабильности, равной « 1 » (линия «главной последовательности»).** По полученным ли-

ниям **пояснить сбалансированность между абстрактностью и нестабильностью** программы. *Пример* **графической иллюстрации приведён в приложении Г.**

5. Сделать выводы по работе, см. **приложение Д.**

1 ВВОДНЫЙ МАТЕРИАЛ

С развитием объектно-ориентированных языков программирования появился **класс метрик, называемый объектно-ориентированными метриками**. Данный вид метрик **направлен на оценку классов внутри программного пакета**.

Метрики Мартина. В категории каких-либо классов**, отдельный класс редко может быть использован повторно изолированно от других классов** [2–4].

Каждый класс имеет, в свою очередь, **группу классов, с которыми он работает совместно и от которых не может быть отделён** простым способом.

Для повторного использования классов необходимо повторно использовать всю группу классов.

Такая группа классов имеет между собой **«сильную» связь (слабое зацепление с иными группами классов) и называется «категорией классов»**.

Для существования «категории классов» имеются условия:

а) все классы в пределах «категории классов» совместно закрыты от изменения. Если один класс должен измениться, все классы в этой категории изменятся с большой вероятностью. Если любой из классов открыт для какой-

либо разновидности изменений, то они все открыты для такой разновидности изменений;

б) классы в «категории классов» повторно используются только вместе. Они сильно взаимозависимы и не могут быть отделены друг от друга. При попытке повторного использования одного класса в категории, все другие классы будут повторно использоваться с ним;

в) классы в «категории классов» разделяют некоторую общую функцию или достигают некоторой общей цели.

По Мартину, **программный пакет должен быть и абстрактным, и стабильным** [4]. То есть, стабильный пакет должен быть и абстрактным, чтобы его стабильность не мешала ему при расширении.

Стабильный пакет не имеет зависимости от других пакетов и полностью **независим**. И, для конкретного пакета с фиксированным содержанием, в него сложно добавить новую функциональность.

Но **если пакет содержит только интерфейс или абстрактный класс** (форму), тогда **будет проще добавить новые функциональные возможности**. **Проще создать новый метод, который добавляет новый функционал к объектам внутри конкретного пакета**, чем добавлять новый метод, который будет работать с каждым из объектов внутри конкретного пакета.

Принцип стабильной абстракции (**SAP** – Stable-Abstraction Principle) **отражает связь между устойчивостью и абстрактностью**. Неустойчивый пакет должен быть конкретным, и его нестабильность позволяет конкретному коду легко внести изменения.

Если **пакет должен быть стабильным, он должен также состоять и из абстрактных классов, чтобы он мог быть расширен**. Пакеты, которые являются стабильными и расширяемыми, являются гибкими и не слишком ограничивают разработку.

Значимость, независимость и стабильность «категории классов» могут быть измерены путём подсчёта зависимостей, которые взаимодействуют с этой категорией. Здесь **определены три метрики**.

1. C_A – **центростремительное сцепление**: это **количество классов вне этой категории, которые зависят от классов внутри этой категории**.

2. C_E – **центробежное сцепление**: это **количество классов внутри этой категории, которые зависят от классов вне этой категории**.

3. Нестабильность:

$$I = C_E/(C_A+C_E). \quad (1)$$

Эта **метрика имеет интервал значений [0; 1]. Если $I = 0$**, то **это указывает на максимально стабильную категорию. Если $I = 1$**, то **это указывает на максимально нестабильную категорию**.

То есть, нестабильность отражает тот факт, что изменения во внешних классах, относительно данной категории, могут привести к необходимости изменений во внутренних классах данной категории.

На **рисунке 1** показана **зависимость нестабильности I от центростремительного C_A и центробежного C_E сцеплений**.

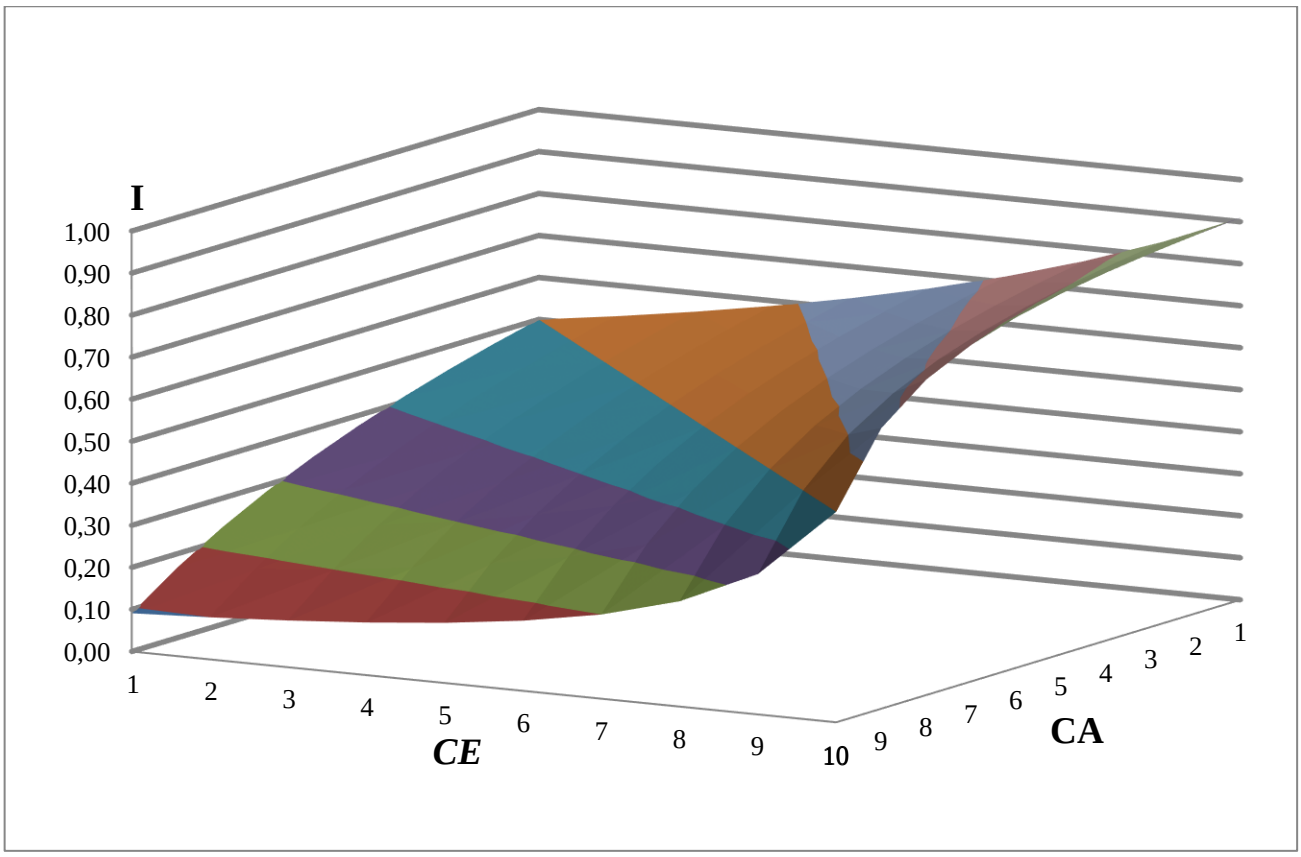


Рисунок 1 – Зависимость **нестабильности от центростремительного и центробежного сцеплений**

4. Также **определена метрика, которая измеряет «абстрактность»** (так как, – **если категория абстрактна, то она достаточно гибкая и может быть расширена**).

Программная система (ПС) с максимально стабильными пакетами будет неизменной. Но **если некоторые части архитектуры будут достаточно гибкие, то ПС будет способна к изменениям и модернизации**. Абстрактные классы могут быть максимально стабильными и достаточно гибкими для расширения. Устойчивые расширяемые пакеты являются гибкими и не ограничивают проектирование [5].

За пределами абстрактного пакета должны быть классы – наследники, которые реализуют недостающие интерфейсы, и поэтому **абстрактный пакет должен иметь потомков**. И не должно быть зависимостей от нестабильных пакетов. Поэтому **нестабильные пакеты не должны быть абстрактными**, а они должны быть конкретными [6].

«Абстрактность» категории определяется следующим образом:

$$A = n_A / n_{All}, \quad (2)$$

где n_A – **количество абстрактных классов (типов) в категории (пакете)**;

n_{All} – **общее количество классов в категории**.

Значения этой метрики **лежат в интервале [0; 1]. Если** $A = 0$, **то категория классов полностью конкретна. Если** $A = 1$, **то категория полностью абстрактна**.

То есть, **понятие «абстрактность категории» совпадает с понятием «абстрактность класса»: показатель абстрактности показывает, какой объём свойств и методов, ещё не имеющих реализации** (написанного кода) **присутствует в данной категории (доля от единицы).**

На рисунке 2 показана связь абстрактности и нестабильности.

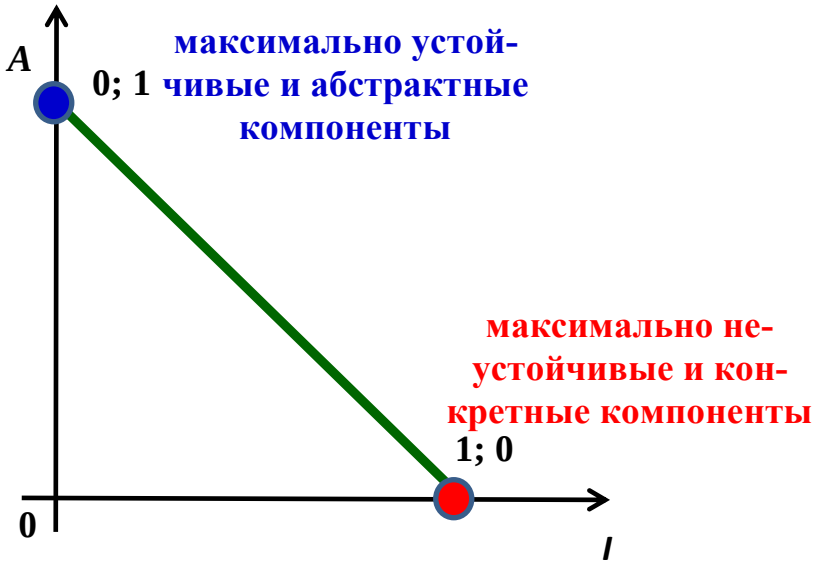


Рисунок 2 – **Связь абстрактности и нестабильности**

На **рисунке 3 показана связь абстрактности и нестабильности через количество абстрактных классов и общее количество классов в категории**.

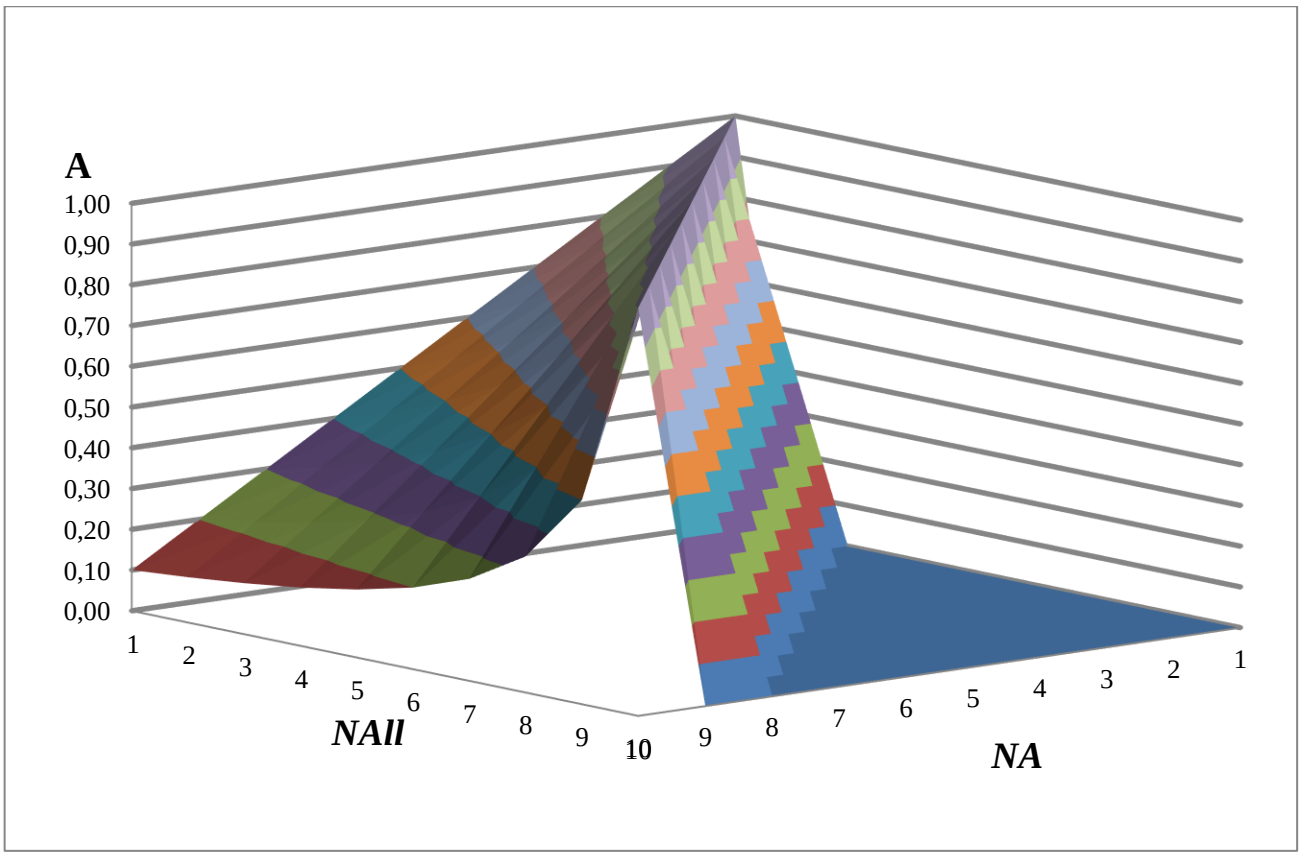


Рисунок 3 – **Связь абстрактности и нестабильности через количество абстрактных классов и общее количество классов в категории**

На основе метрик Мартина **можно построить график, на котором будет связь между абстрактностью *A* и нестабильностью *I*** (рисунок 2)**.**

На графике можно **выделить точки, заданные суммой**: $I + A = 1$ **(лежат на прямой**).

На этой **прямой лежат категории классов, имеющие наилучшую сбалансированность между абстрактностью и нестабильностью**. Такая **прямая называется «главной последовательностью»**.

Компоненты ПС имеют разную абстрактность и нестабильность (*например*, если один абстрактный класс наследует другой абстрактный класс, то получается новый абстрактный класс, имеющий зависимость). То есть, несмотря на абстрактность, класс не может быть максимально устойчивым.

На линии главной последовательности лежат точки, отражающие оптимальные позиции для компонентов. **Также** можно **определить области, где компоненты не должны находиться** («**зоны исключения**»), **рисунок 4**.

«**Зона боли**» – это **устойчивые и конкретные компоненты, и их не расширить из-за неабстрактности, и сложно изменить из-за большой устойчивости**. Обычно это **схемы баз данных**, которые **максимально конкретны и имеют огромное число связей**. В этой зоне находится **и конкретная библиотека вспомогательных функций**. **Хотя** для неё ***I* = 1**, но **зачастую отдельный компонент, хотя его классы конкретны, используется очень широко, и его изменение ведёт к беспорядочности**. Это **негибкие компоненты, вызывающие** «**боль**».

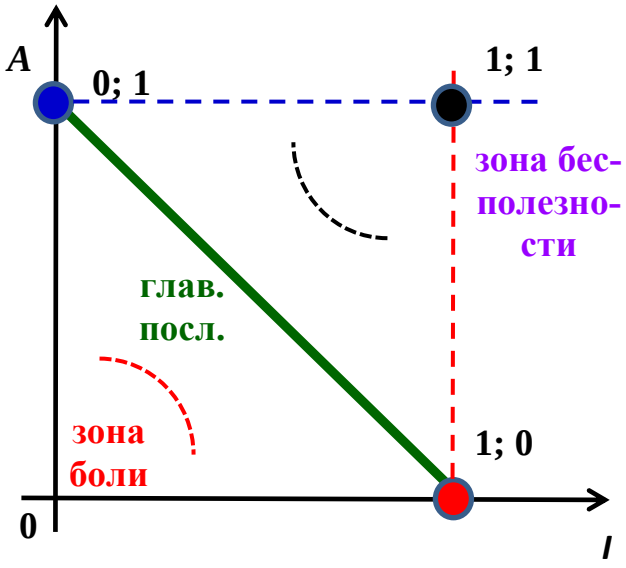


Рисунок 4.4 – **Связь абстрактности и нестабильности**

«**Зона бесполезности**» – это **максимально абстрактные компоненты, не имеющие входящих зависимостей**, и **они бесполезны**. **Обычно это оставшиеся абстрактные классы, которые не были реализованы** («осколки»). Присутствие этих бесполезных сущностей нежелательно.

Не все компоненты находятся на главной последовательности. В зависимости от сочетания нестабильности и абстрактности компонента, то есть компонент определяется координатой точки **(*I*; *A*)** по типу рисунка 4, можно оценить расстояние от указанной точки до линии «главной последовательности».

Для этого определены **ещё две метрики, из которых видно, что чем ближе к главной последовательности находятся категории, тем лучше: ближе к условию сбалансированности абстрактности** A **и устойчивости** I.

5. Расстояние до главной последовательности:

$$D = \frac{|(A+I-1)|}{\sqrt{2}}. \quad (3)$$

Метрика **изменяется в интервале [0; 0,707]**. Для удобства пользования Мартин предложил следующую **нормализованную метрику, изменяющуюся в интервале [0; 1]**.

6. Нормализированной расстояние до главной последовательности:

$$D_n = |A + I - 1|. \quad (4)$$

Нормализованное расстояние от главной последовательности **характеризует близость класса к главной последовательности, для которой выполняется условие сбалансированности абстрактности** A **и устойчивости** I. Если эта метрика равна нулю, то пакет находится на линии главной последовательности. Если эта метрика равна единице, то пакет максимально далеко от линии главной последовательности.

Компонент со значением метрики D_n **далеким от нуля требует пересмотра**.

Если выполнить статистический анализ разработки, то **можно найти среднее и дисперсию всех метрик** D_n для компонентов в архитектуре.

Если среднее и дисперсия близки к нулю, значит весь проект находится рядом с главной последовательностью. Для **дисперсии можно установить доверительный интервал и рассматривать выход за его границы как появление компонентов, оказавшихся в зоне исключения** в сравнении с остальными.

ПРИЛОЖЕНИЕ А

Графическое представление схемы алгоритма программы

1) представить рисунком схему алгоритма программы, выполненной на знакомом языке программирования (одном языке).

Размер листинга не более одной–двух страниц формата А4, индивидуален для каждого студента.

Пример программы алгоритма схемы приведён на рисунке А.1, [3].

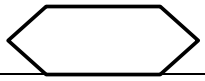
Схема алгоритма программы должна отражать последовательность операций в программе, а не показывать схему работы какой-то системы.

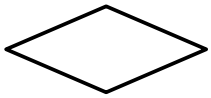
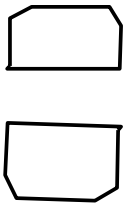
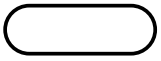
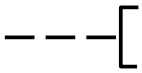
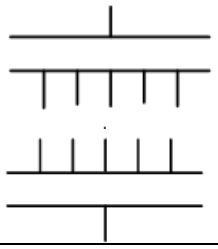
Не использовать синтаксис языка программирования, и при необходимости руководствоваться п. 4.1.4 ГОСТ [3] («Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ»).

Также, п. 4.1.5 ГОСТ [3]: («В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом»).

Условные графические обозначения (УГО) и правила выполнения по ГОСТ – краткая справка в таблице А.1.

Таблица А.1 – УГО в схемах алгоритмов, программ, данных и систем

УГО	Наименование	Назначение
	**Данные**	Определяет ввод или вывод на внешнее устройство или носитель данных.
	**Процесс**	Отражает обработку данных (выполнение операции, группы операций).
	**Типовой процесс**	Отображает предопределенный процесс, состоящий из одной или нескольких операций программы, которые определены в другом месте.
	**Подготовка**	Отображает модификацию параметра для управления циклом со счетчиком.

	**Решение**	Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два или более альтернативных выходов, один из которых активизируется после вычисления условия.
	**Граница цикла**	Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла.
	**Соединитель**	Используется для обрыва линии и продолжения ее в другом месте.
	**Знак завершения**	Определяет начало и конец структурной схемы алгоритма.
	**Комментарий**	Используется для добавления пояснительных записей.
	Основная линия	Отражает последовательность выполнения действий в алгоритме.
	начало **Параллельные действия** конец	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами.

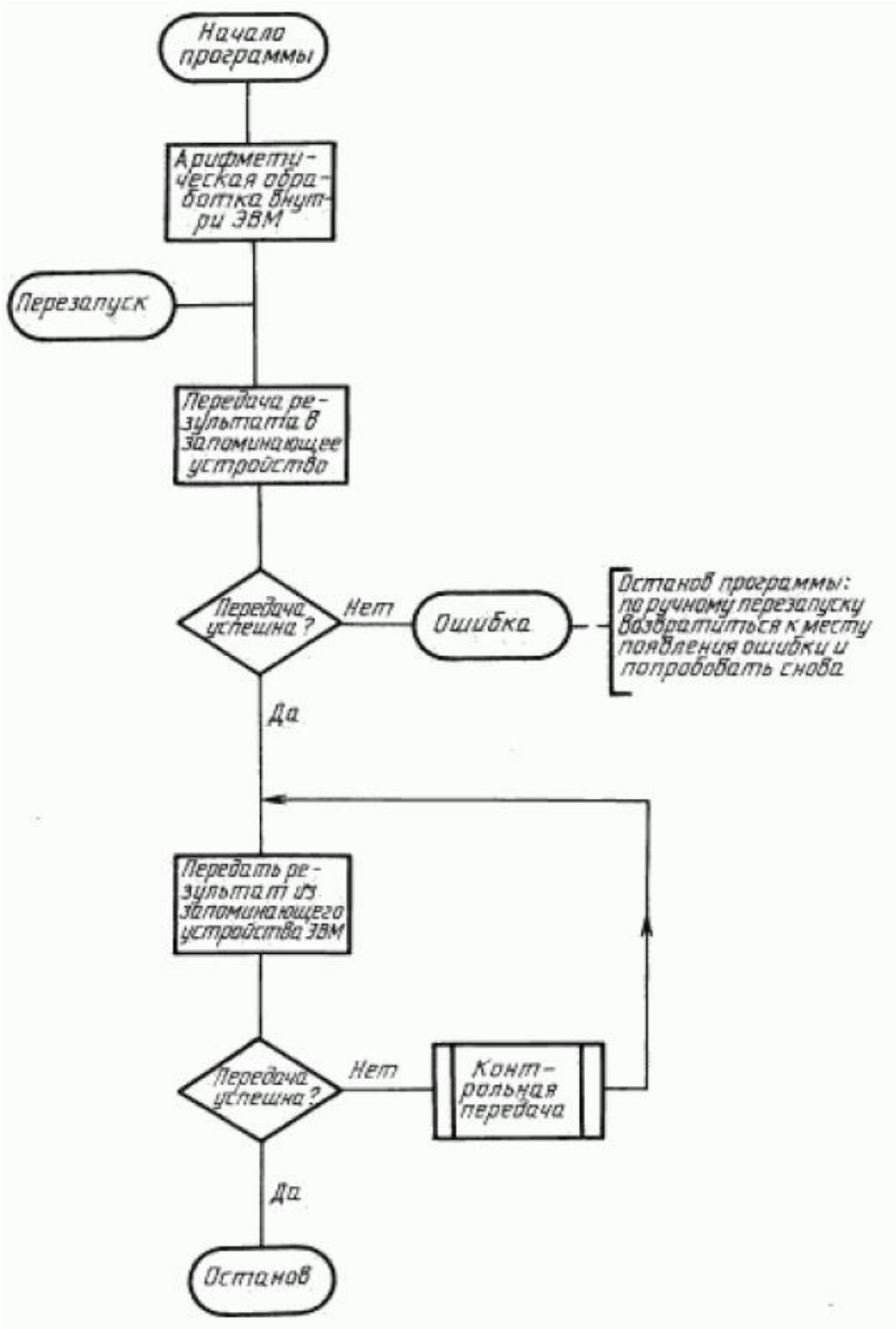


Рисунок А.1 – Пример схемы программы (алгоритма)

ПРИЛОЖЕНИЕ Б

Построение «диаграммы классов» и «диаграммы обзора взаимодействия» исследуемой программы

Проанализировать листинг, опираясь на синтаксис языка. Листинг приводить в приложении к основной части отчёта по ПЗ3, а не в теле его основной части.

Пример «диаграммы классов» и её краткого описания для некой абстрактной программы приведен **на рисунке Б.1**.

1. Book – класс описывает сущность книги. **Атрибуты:**

– title: str – название книги,

– author: str – автор книги,

– year: int – год издания,

– available: bool – флаг доступности (доступна или выдана).

Методы:

– init_– конструктор для создания экземпляра книги,

– repr_– возвращает строковое представление книги.

2. AbstractStorage (абстрактный класс) – определяет интерфейс для работы с хранилищем книг. **Методы:**

– add(book: Book) – добавление книги,

– search_by_title(title_query: str) – поиск по названию,

– search_by_author(author_query: str) – поиск по автору,

– list_all() – вывод всех книг.

3. Catalog – реализация абстрактного хранилища (AbstractStorage). **Атрибуты:**

– books: List[Book] – список книг в каталоге.

Методы:

– реализует интерфейс AbstractStorage: добавление, поиск по названию, поиск по автору, вывод всех книг.

4. Library – сервисный слой, использующий хранилище (AbstractStorage) для предоставления бизнес-функций. **Атрибуты:**

– catalog: AbstractStorage – ссылка на используемое хранилище книг.

Методы:

– add_book(...) – добавление книги,

– find_by_title(title: str) – поиск по названию,

– find_by_author(author: str) – поиск по автору,

– all_books() – вывод всех книг.

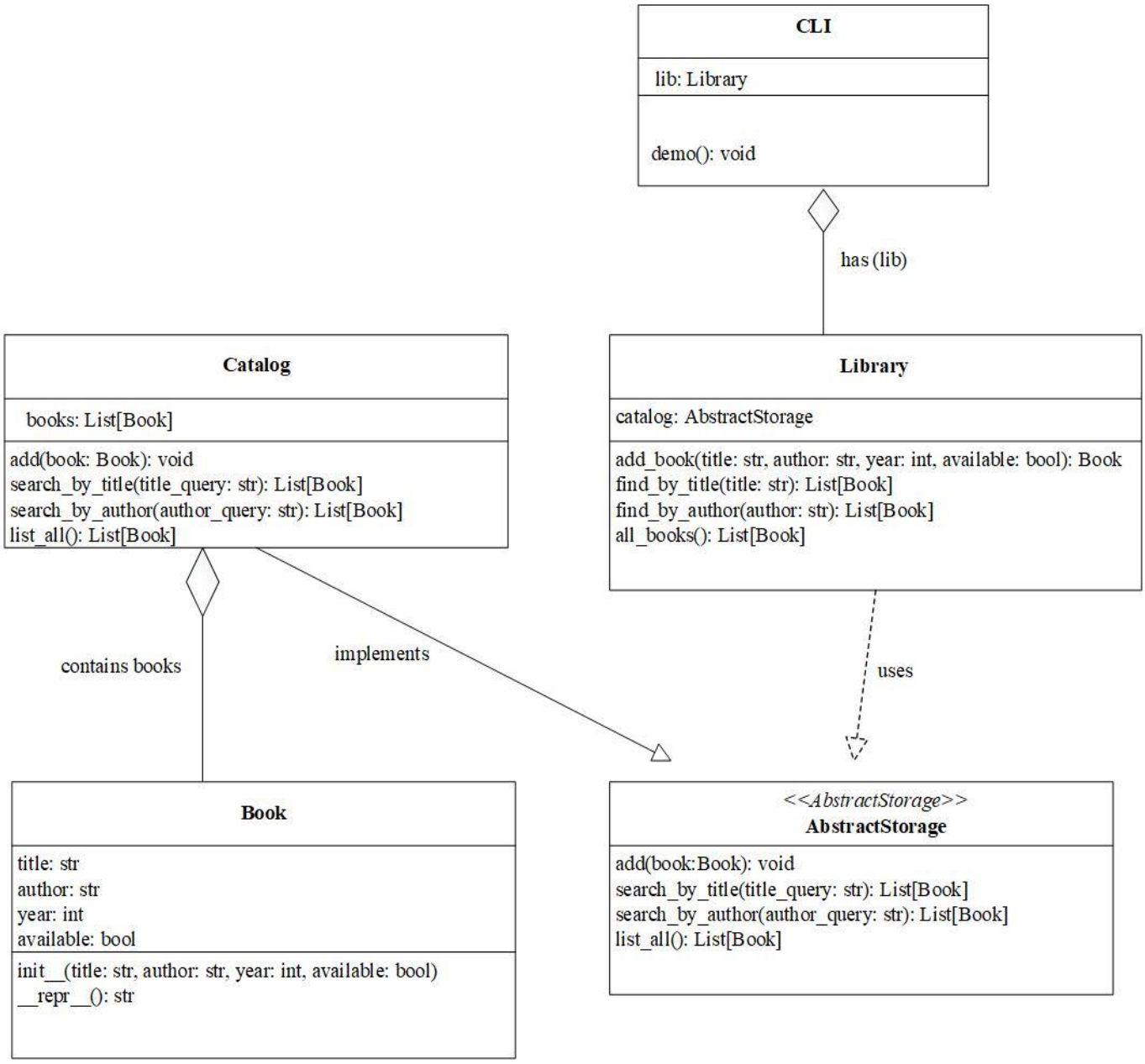


Рисунок Б.1 – **Пример «диаграммы классов» программы**

5. CLI – класс пользовательского интерфейса для демонстрации возможностей библиотеки. **Атрибуты:**

lib: Library – ссылка на объект библиотеки.

Методы:

– demo() – демонстрационный сценарий работы (добавление книг, вывод, поиск).

Связи классов:

– Catalog реализует интерфейс AbstractStorage;

– Library использует абстракцию AbstractStorage (зависимость);

– CLI агрегирует объект Library (содержит ссылку на него);

– Catalog агрегирует объекты Book (хранит коллекцию книг).

ПРИЛОЖЕНИЕ В
Пример расчёта метрик Мартина

Пример **расчёта метрик Мартина для некой программы** приведён ниже. Использована диаграмма классов, показанная на рисунке Б.1. Определены следующие метрики.

1. C_A – **центростремительное сцепление**: **количество классов вне данного класса, которые зависят от данного класса**.

2. C_E – **центробежное сцепление**: **количество классов**, от которых зависит данный класс.

3. **Нестабильность**: $I = C_E/(C_A+C_E)$. Лежит в диапазоне [0; 1]. Ноль – абсолютно стабильный класс, единица – абсолютно нестабильный класс.

4. **Абстрактность категории классов**: $A = n_A/n_{All}$, где n_A – **количество абстрактных классов в категории классов**; n_{All} – **общее количество классов в категории классов**.

5. **Нормализированной расстояние до «главной последовательности»** категории классов:

$$D_n = |A + I - 1|. \quad \text{(В.1)}$$

1. Для класса **Book**.

Зависимостей от других классов нет, и центробежное сцепление $C_E = 0$.

От данного класса зависит один класс Catalog, поэтому центростремительное сцепление $C_A = 1$.

Нестобильность этого класса равна:

$$I = C_E/(C_A+C_E) = 0 / (1 + 0) = 0. \quad \text{(В.2)}$$

Класс **Book** является полностью стабильным.

2. Для класса **AbstractStorage**.

От него зависят класс Catalog и класс Library. Поэтому $C_A = 2$.

Так как этот класс AbstractStorage сам ни от кого не зависит, то $C_E = 0$.

Отсюда, нестабильность данного класса равна:

$$\boldsymbol{I = C_E/(C_A+C_E)} = 0 / (2 + 0) = 0. \quad \text{(В.3)}$$

Класс **AbstractStorage** является полностью стабильным.

3. Для класса **Catalog**.

Этот класс зависит от класса Book и он связан (реализует) с классом AbstractStorage, и тогда $\boldsymbol{C_E} = 2$.

Также от этого класса Catalog зависит класс Library (использует хранилище), и поэтому $\boldsymbol{C_A} = 1$.

Тогда, нестабильность данного класса равна:

$$\boldsymbol{I = C_E/(C_A+C_E)} = 2 / (1 + 2) = 0{,}67. \quad \text{(В.4)}$$

Наблюдается примерно средняя устойчивость данного класса.

4. Для класса **Library**.

Здесь класс Library зависит от класса AbstractStorage, и поэтому $\boldsymbol{C_E} = 1$.

Кроме этого, от этого класса Library зависит класс CLI, что ведёт к тому, что $\boldsymbol{C_A} = 1$.

Определим нестабильность класса Library равной:

$$\boldsymbol{I = C_E/(C_A+C_E)} = 1 / (1 + 1) = 0{,}5. \quad \text{(В.5)}$$

Получили среднюю устойчивость данного класса.

5. Для класса **CLI**.

Этот класс зависит от класса Library, и следовательно, $\boldsymbol{C_E} = 1$.

Также, от класса CLI не зависят другие классы, и $\boldsymbol{C_A} = 0$.

Тогда нестабильность класса CLI равна:

$$\boldsymbol{I = C_E/(C_A+C_E)} = 1 / (0 + 1) = 1{,}0. \quad \text{(В.6)}$$

Получилось, класс CLI полностью нестабильный.

6. Абстрактность рассматриваемой программы будет равна:

$$\boldsymbol{A = n_A/n_{All} = 1 / 5 = 0{,}2}. \quad \text{(В.7)}$$

Для построения линии «**главная последовательность**» не целесообразно брать нестабильность программы как среднее арифметическое нестабильности классов. Поскольку у нестабильности классов наблюдается резкий разброс значений (от нуля до единицы), можно предполагать, что нестабильность классов, как случайная величина, не подчиняется нормальному закону распределения вероятности появления тех или иных значений нестабильности классов.

Поэтому нужно принять нестабильность программы равной наибольшему зна-

чению $\boldsymbol{I}$ = 1,0, что выявлено для класса CLI.

7. Нормализированной расстояние до «главной последовательности» исследуемой программы определится:

$$\boldsymbol{D_n = |A + I - 1| = |0,2 + 1 - 1| = 0,2.} \quad \text{(В.8)}$$

Для визуальной оценки отклонения от линии «главной последовательности» нужно будет построить соответствующие графики. Итоговая сводная таблица, включающая найденные значения метрик Мартина – это таблица В.1.

Таблица В.1 – Сводная таблица найденных значений метрик Мартина

Класс (программа)	**Центробежное сцепление C_E класса**	**Центростремительное сцепление C_A класса**	**Нестабильность I**	**Абстрактность A**	**Расстояние D_n**	**Комментарий**
Book	ноль	1	ноль	–	–	Полностью стабилен.
Abstract Storage	ноль	2	ноль	–	–	Полностью стабилен.
Catalog	2	1	0,67	–	–	Средняя устойчивость
Library	1	1	0,5	–	–	Средняя устойчивость
CLI	1	0	1	–	–	Полностью нестабильный
Программа	–	–	1	0,2	0,2	–

ПРИЛОЖЕНИЕ Г

Пример графической иллюстрации сбалансированности программы по мерам Мартина «нестабильность» и «абстрактность»

Нормализованное расстояние D_n до линии «**главной последовательности**» – это мера, отражающая, насколько близко данная программа расположен к идеальной линии баланса между абстрактностью и нестабильностью (устойчивостью).

Если значение меры равно нулю, линия сбалансированности программы будет лежать непосредственно на лини «главной последовательности», что будет соответствовать идеальной сбалансированности. Если значение меры равно единице, то это значит, что линия сбалансированности программы максимально удалена от линии идеального баланса.

Графики, **отражающие связь абстрактности и нестабильности – линии**

идеального и реального баланса – приведены на **рисунке Г.1**.

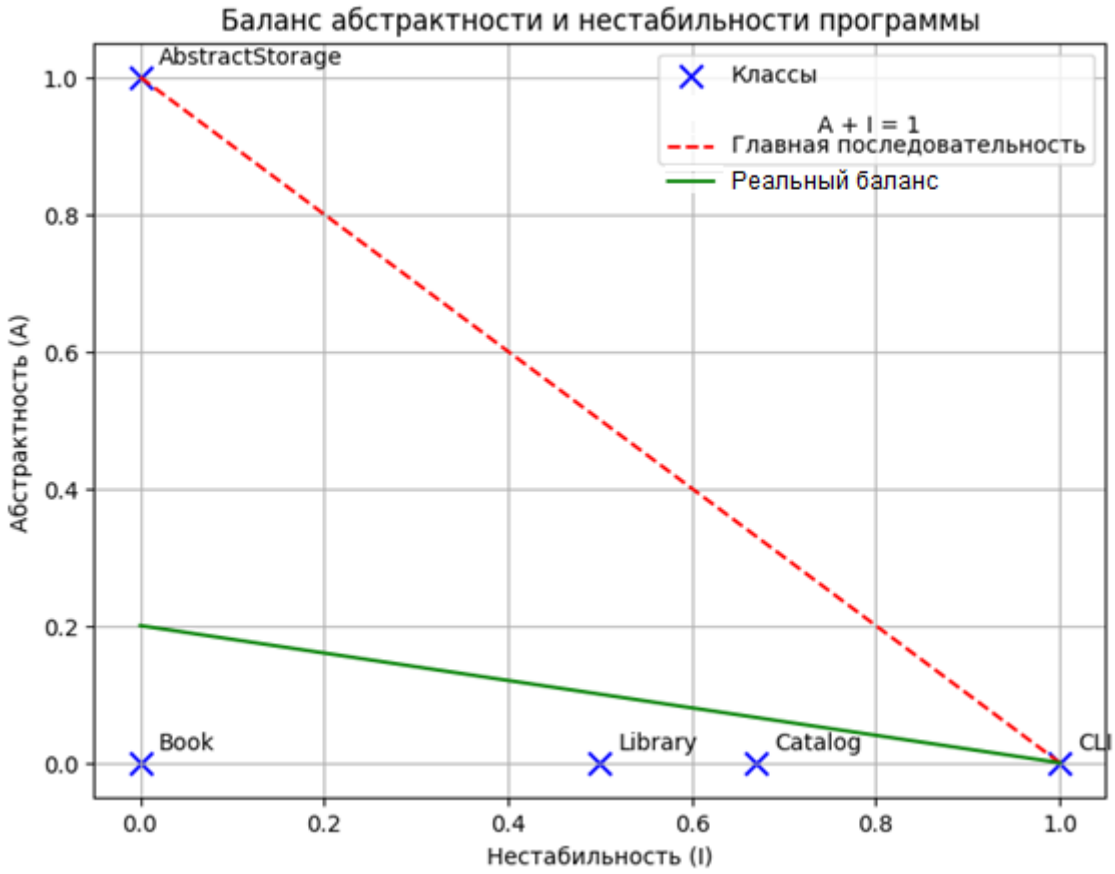


Рисунок Г.1 – Графики идеальной и реальной сбалансированности программы по мерам Мартина «абстрактность» и «нестабильность»

Графики на рисунке Г.1 и полученные значения мер абстрактности и нестабильности Мартина показывают, что большинство классов программы расположены близко к «Зоне боли» – области, где классы программы в среднем стабильны и конкретны, то есть практически не используют абстракции и имеют много входящих зависимостей. Такой дисбаланс затрудняет расширение и модификацию программы в будущем.

Линия связи абстрактности и нестабильности программы (линия реального баланса) излишне далеко отстоит от линии «главной последовательности», потому что классы либо слишком конкретны и в среднем стабильны, и недостаточно абстрактны, чтобы обеспечивать гибкость структуры программы.

Для улучшения баланса рекомендуется выделять больше абстракций для стабильно используемых компонентов и обеспечить, чтобы конкретные классы не становились точками жёсткой связности программы.

ПРИЛОЖЕНИЕ Д

Выводы по расчёту мер Мартина для программы, созданной в парадигме ООП

В результате выполнения расчетной работы произведена оценка качества программы с помощью объектно-ориентированных метрик Мартина. Структура программы содержит как абстрактные, так и конкретные классы, реализовано наследование и композиционные связи между компонентами, что подтверждается диаграммой классов.

Анализ полученных результатов показывает, что архитектура системы формируется преимущественно за счет конкретных компонентов, что обеспечивает абстрактность программы в 0,2, а нестабильность – в единицу. Это свидетельствует, что большая часть классов реализует конкретную логику и слабо связана с абстракциями, а сами компоненты программы нестабильны и могут изменяться без существенного влияния на её другие части.

Для повышения масштабируемости и гибкости рекомендуется увеличить долю абстрактных интерфейсов и усилить абстрагирование в ключевых компонентах, что позволит упростить их дальнейшее развитие и повторное использование.

В программе особое внимание нужно уделить классам Library и Catalog, поскольку в них сосредотачивается основная бизнес-логика, и именно здесь могут появляться длинные или сложные методы. Их оптимизация – это декомпозиция функций, введение новых вспомогательных классов, строгое разделение ответственностей, что позволит повысить читаемость и поддерживаемость всей архитектуры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.701− 90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 1990. – 8 с.

2. Объектно-ориентированное программирование. – https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5 (дата обращения 25.04.2023).

3. Понятие класса. – https://studbooks.net/2181612/informatika/ponyatie_klassa (дата обращения 25.04.2023).

4. Подбельский В. В. Виртуальные функции и абстрактные классы. – М.: Финансы и статистика, 2003. – 560 с.

5. Robert C. Martin, Agile Software Development, Pearson Education Inc, 2003.

6. Robert C. Martin, Object Oriented Design. Quality Metrics an Analysis of Dependencies. – http://www.objectmentor.com/resources/articles/oodmetrc.pdf , (дата обращения 2004-06-04).