ПЗ 3. Метрики сложности потока данных. Метрика Чапина.

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

ЭЛЕКТРОННЫЙ ВАРИАНТ

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

Тематика: сложность потока данных программы. Метрика Чапина.

ЗАДАНИЕ

1. Выбрать программу, в которой ОБЯЗАТЕЛЬНО должны быть ВЕТВЛЕНИЯ, или ЦИКЛЫ (или то и другое).

Рекомендуется выбрать программу, выполненную в парадигме объектно-ориентированного программирования (ООП). **В** ином случае, **возможно использование программы, выполненной в парадигме императивного программирования** (ИП).

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**.

Парадигма ИП: стиль написания кода, **основанный на последовательности команд, которые явно указывают как шаг за шагом выполнять вычисления**. В ИП описывается процесс (**что делать**), **изменяя состояние программы** с помощью переменных и инструкций.

Последовательность инструкций означает, что **команды кода выполняются в строгом порядке одна за другой, программа работает изменением значений переменных в памяти. Эта парадигма предоставляет полный контроль над способом выполнения задачи и ассоциируется с рецептом приготовления блюд, когда каждое действие чётко описано.**

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

2. Выполнить фрагментацию программы.

2.1. Если выбрана программа, созданная в парадигме ООП, 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.**

2.2. Если выбрана программа, созданная в парадигме ИП, то

представить схему работы программы в виде «диаграммы обзора взаимодействия» связанных между собой фрагментов программы в виде модулей, блоков или отдельных небольших частей (функций).

При фрагментации программы обратите внимание на основные атрибуты блоков:

1) сколько функций выполняет блок;

2) какой логикой обладает блок;

3) в скольких штук контекстов (содержательных частях) используется блок.

Построить **«диаграмму обзора взаимодействия» можно теми же средствами, что указаны в п. 2.1.**

Примеры **построения «диаграммы классов» и «диаграммы обзора взаимодействия» некой абстрактной программы** приведены **в приложении Б**.

3. Провести классификацию «ролей переменных» по методике Чапина в исследуемой программе. Результат классификации представить в табличном виде. *Пример* **частичной классификации «ролей переменных» для некой программы** приведён **в приложении В.**

4. Провести расчёты значения меры сложности Q исследуемой программы по методу Чапина. *Пример* **методики расчёта приведён в приложении Г (частично).**

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

1 ВВОДНЫЙ МАТЕРИАЛ

Метрика Чапина относится к классу метрик «**сложности потока данных**» [1].

Данный класс метрик **устанавливает сложность структуры программы как на основе количественных подсчётов, так и на основе анализа управляющих структур**.

Суть метода состоит **в оценке информационной прочности отдельно взятого программного модуля (блока, фрагмента программы) с помощью анализа характера использования переменных из списка ввода и вывода программы**.

Метрика позволяет определить **численное выражение сложности понимания функции, выполняемой программным продуктом.** Автор метрики указал, что «Функция ПО – это то, что «ПО управляет компьютером». То есть, **функция ПО определена взаимосвязями между свойствами ПО, а не тем, что эти свойства собой представляют**.

Метрика **выясняет организацию и взаимосвязи элементов ПО**, а не выявляет конкретные механизмы алгоритма, реализованного в программном модуле. Чапин утверждает, что **значение сложности ПО определяется** в большей степени **не областями и диапазонами входных и выходных данных, а тем, как компоненты наборов входных и выходных данных связаны между собой**. Распознав роль данных, можно установить спецификацию функции ПО.

Всё **множество переменных**, составляющих список ввода и вывода (программы), разбивается на четыре функциональные группы.

1. ***P*** – **входные данные, которые необходимы функции для обработки других входных данных с целью создания выходных данных –** «данные роли «***P***» («данные ***P***»). Эти переменные не модифицируются и содержат обрабатываемую информацию, или указывают на правила её обработки. Данные ***P*** – это входные данные, используемые программой [2].

2. ***M*** – данные, которые **изменены, модифицированы по значению или их признакам, а также вновь созданы в программе** в результате выполнения функции, **и являются её выходными данными**.

3. *C* – переменные, которые **определяют выбор действий в программе или задают условия принятия решения** на исполнение функций, им отведена контролирующая роль. Такие переменные предназначены для передачи управления, изменения логики вычислительных процессов и т. д.

4. *T* – данные, которые **используются функцией ПО, но при этом не меняют значения или признаки** (тип) **каких-либо переменных**. *Например*, функция программы заключается в **передаче данных без изменений**, то есть в **общении между отдельными частями ПО или между отдельными программными модулями**. Это «данные роли «***T***» («данные ***T***») – «**передаваемые и обеспечивающие**».

Такие переменные могут использоваться для выполнения промежуточных действий или играют обеспечивающую роль в программе.

Признаки данных *T*: **а)** данные **используются функцией ПО, при этом не изменяя значений** или характера (типа) **переменных**; **б) обеспечивают трансляцию** (общение) **между программными модулями**. Часто ошибочно указывают на отсутствие этих данных, **отбрасывая их свойство как обеспечивающих работу программы и реализацию функции ПО**.

В качестве данных ***T*** **ошибочно не учитываются директивы и утилиты**, обеспечивающие работу программы. Н. Чапин указывал, что его метод – это «**численное выражение сложности понимания людьми функции, выполняемой программным продуктом**». **Сложность понимания программы увеличивается с использованием**:

– **внешних и типовых библиотек**;

– **программного интерфейса приложения** (**API**) операционной системы;

– **специфичных заголовочных файлов языков программирования**, в которых объявляются функции, предоставляющие интерфейс доступа к API. Так используется функционал, предоставляемый операционной системой (ОС).

Поэтому, для данных группы ***T*** в метрике Чапина нужно применять термин «**передаваемые и обеспечивающие**».

Поскольку каждая переменная может выполнять одновременно

несколько ролей, необходимо учитывать её в каждой соответствующей ролевой группе.

Влияние на сложность программы каждой роли данных **учтено соответствующими весовыми коэффициентами**. Данные ***C***, являясь управляющими, **вносят наибольший вклад в сложность**. А данные ***M***, поскольку **это изменённые, созданные или модифицированные по значению или типу в результате выполнения функции, и они являются выходными данными** – это основные участниками в формировании сложности.

Данные ***P*** **вносят некоторую небольшую** сложность. Данные ***T***, **выступающие как «передаваемые или обеспечивающие»**, которые только **передаются без изменений** между отдельными частями ПО **или выступают как** данные, которые также **обеспечивают за счёт функции программы и передачу данных**, они **в наименьшей степени увеличивают сложность** программного продукта.

Чапин показал, что если **длительность или объём итерации для выхода из цикла определяются данными *C***, существующими **вне конкретного модуля**, **то** сложность ***Q*** **нелинейно растёт**. Сложность Q_i **отдельного *i*-го программного модуля** (процедуры, фрагмента, блока) программы, **состоящей из нескольких модулей**, определяется формулами (1)–(3):

$$Q_i = \sqrt{R_i \cdot W_i}, \quad (1)$$

где R_i – **фактор повторения**, **отражающий увеличение сложности** из-за роста длительности или объёма итераций для выхода из цикла вследствие наличия данных ***C*** **вне данного модуля**:

$$R_i = 1 + \left(\frac{E_i}{3}\right)^2, \quad (2)$$

где E_i – величины, отражающие **число процедур, содержащих данные *C***, определяющие выход из цикла. Определение E_i показано **ниже**;

W_i – **взвешенная сумма количеств элементов данных *T, P, M* и *C***, учитывающая весовые коэффициенты:

$$W_i = 0,5T_i + P_i + 2M_i + 3C_i. \quad (3)$$

Если данные ***C*** **являются константами или поступают только из собственно самой процедуры** (блока), **то циклы или итерации игнорируются**, поскольку **они выполняются полностью внутри модуля без итеративно вызываемых подчинённых модулей** (процедур). Для каждого из таких данных ***C*** **не прибавляется никаких констант**, т. е. $E = 0$, а **фактор повторения** R_i **равен единице**.

Если данные ***C*** используются **из тела подчинённого цикла** (по сути, вложенного цикла), **для каждого из таких элементов данных** ***C*** **прибавляется единица**, т. е. $E = 1$, а **фактор повторения** R_i равен **1,1(1)**.

Когда данные ***C*** берутся **извне относительно тела цикла**, **то добавляют константу** «**два**» для каждого из таких элементов данных ***C***, т. е. $E = 2$, а **фактор повторения** R_i равен **1,4(4)**.

То есть, **если элемент данных** ***C*** **инициализируется значением вне цикла и** $E = 2$, и в то же время **этот элемент данных** ***C*** **изменяется внутри тела цикла**, то $E = 2 + 1 = 3$, а **фактор повторения** R_i равен **2,0**.

Вычислением Q_i по формуле (1) получают меру сложности отдельного ***i*-го модуля** или блока кода. **Значение сложности программы** ***Q*** находится **по формуле (4) путём определения арифметического среднего значений сложности** Q_i отдельных модулей (блоков), составляющих программу из ***n*** компонент:

$$Q = \frac{\sum_i Q_i}{n}. \quad (4)$$

Для рационально спроектированной программы предполагается примерно равномерное распределение сложности по отдельным процедурам (блокам).

Исследования Н. Чапина показали, что **нижняя граница сложности программного модуля или блока обычно равна единице**.

Сложность, **равная нулю**, будет у модуля без данных ***T***, ***P***, ***C*** или ***M***. Такой модуль без функций и с отсутствием данных ***T***, объявить проблематично.

На основе эмпирических данных оценки сложности ***Q*** ПО, Чапин показал, что верхней границы для ***Q*** нет, **но обычно она не превышает значения 11 для процедур (блоков) с ветвлением**. Большие значения нетипичны для структурного программирования.

Значение *Q* редко превышает 5 для процедур без ветвления. Это листовые модули дерева структуры программной системы.

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.

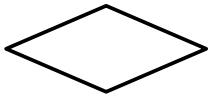
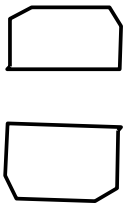
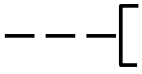
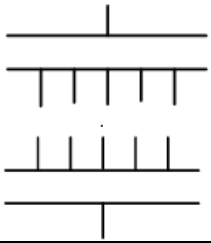
Не использовать синтаксис языка программирования, и при необходимости руководствоваться п. 4.1.4 ГОСТ [3] («Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, текст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

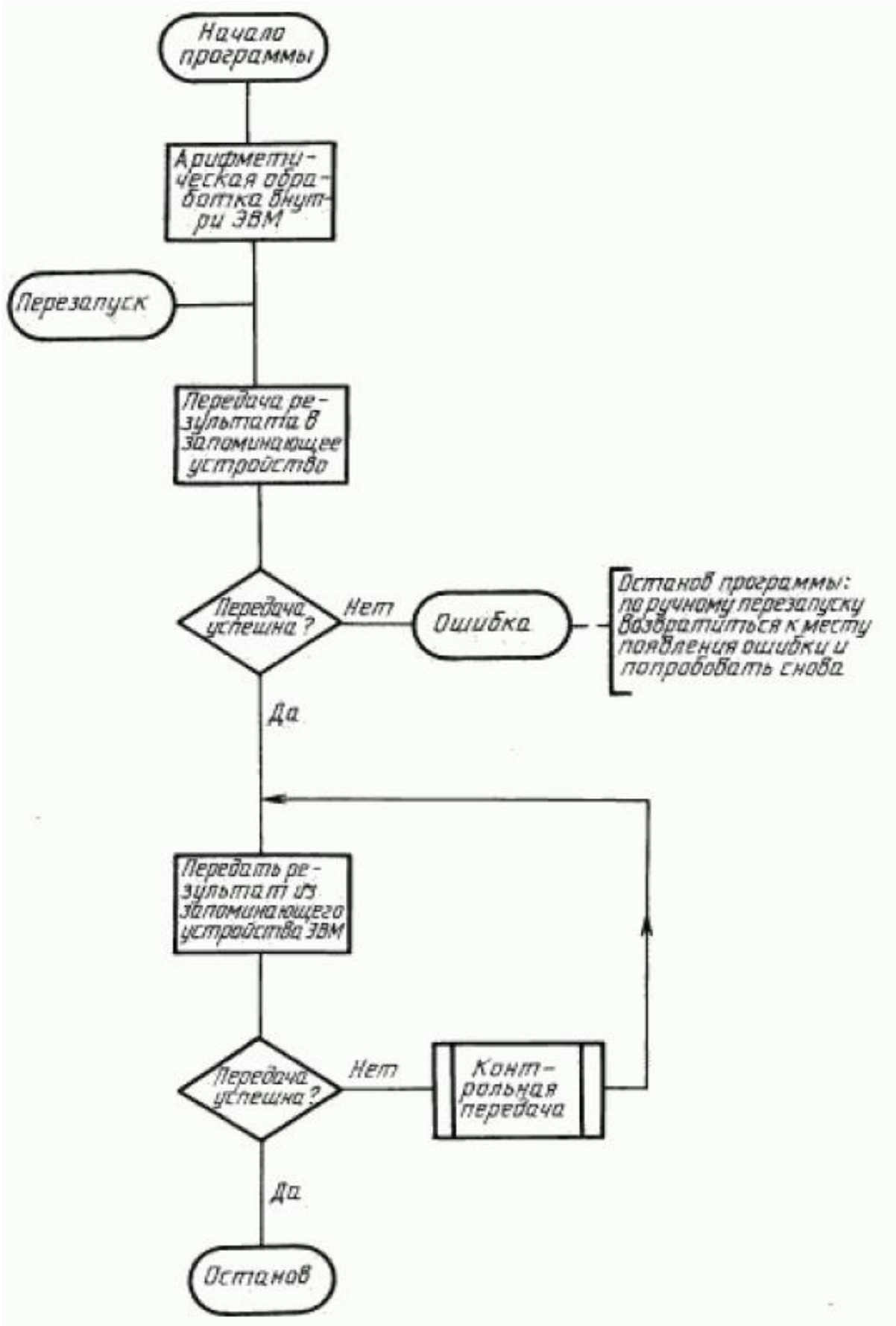


Рисунок А.1 – Пример схемы программы (алгоритма)

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 ПЗ, 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.

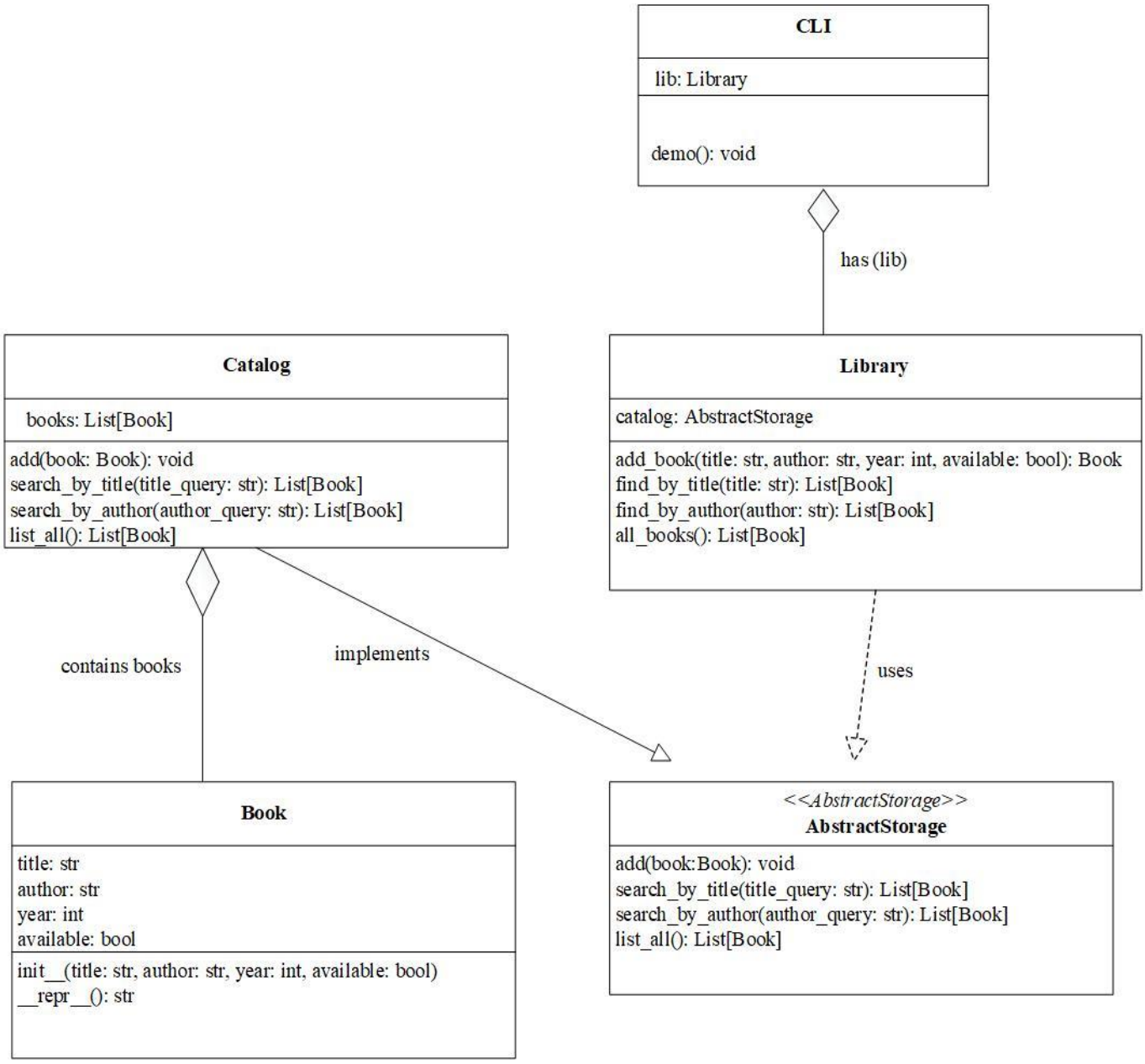


Рисунок Б.1 – **Пример «диаграммы классов» программы**

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).

Ниже дан **пример частичной «диаграммы обзора взаимодействия» и её краткого описания для некой абстрактной программы**, а вид диаграммы приведён **на рисунке Б.2**.

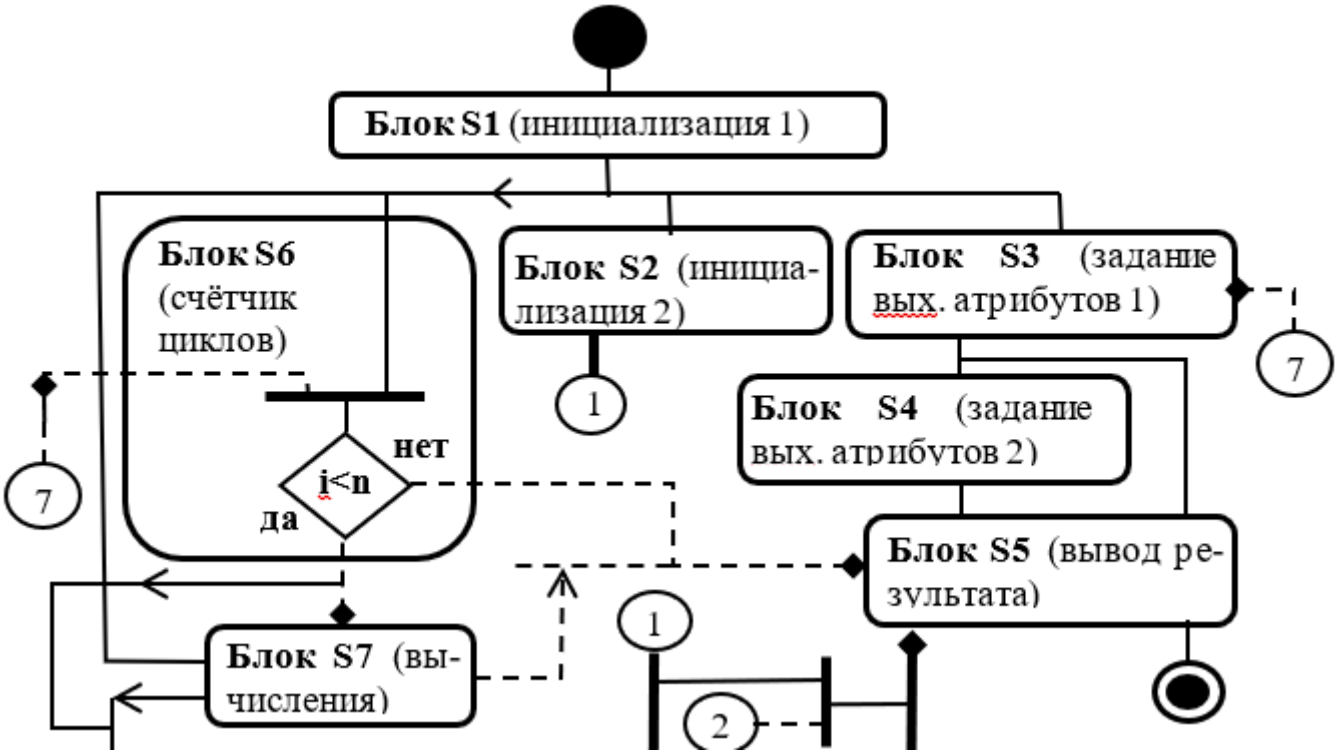


Рисунок Б.2 – **Пример частичной «диаграммы обзора взаимодействия» исследуемой программы**

Блок S1 предназначен для очистки экрана, установки локалей и получения указателя на дескриптор стандартного вывода. Здесь находится переменная «дескриптор hstdout» и указаны заголовочные файлы ОС и библиотеки языка.

Блок S2 предназначен для установки курсора в начальную позицию с координатами (X, Y) и задания pi, а также назначения строк шапки таблицы. Проводится инициализация переменных a=0 и x_0=a, а также хранятся начальные минимальное и максимальное значения функций $F_{1НАЧ\ MIN}$, $F_{2НАЧ\ MIN}$, $F_{1НАЧ\ MAX}$ и

$F_{2НАЧ\ MAX}$. В блоке назначается количество итераций n для поиска значений обеих функций, в том числе инициализации параметров (b и dx) для поиска значений функций в цикле.

Блок S3 предназначен для назначения строки нижней части таблицы.

Блок S4 предназначен для задания выходных атрибутов, определяя цвет текста. Здесь находится переменная «дескриптор hstdout».

Блок S5 выводит результаты расчёта – в нём выводятся переменные: min1; max1; max2; min2. Значения переменных выводятся по окончании выполнения всех циклов.

Блок S6 – счётчик циклов для установки начального значения счётчика циклов i=0, и содержит переменную i текущего значения количества циклов. Блок активирует цикличность работы при условии i<n, обеспечивая поиск значений min1 при условии «$F1_i$ меньше min1»; max1 при условии «$F1_i$ больше max1»; min2 при условии «$F2_i$ меньше min2»; max2 при условии «$F2_i$ больше max2».

Блок S7 – вычисляются величины $F1_i$ и $F2_i$ при значении аргумента x_i в текущем цикле. Номер последующего цикла (i+1). Также выводится строка таблицы, и затем значения величин $F1_i$, $F2_i$, x_i и (i+1) выводятся в строку таблицы.

ПРИЛОЖЕНИЕ В

Пример классификации «ролей переменных» по методике Чапина

Пример **частичной классификации ролей переменных в блоках некой программы** приведён в таблице В.1.

Таблица В.1 – Классификация переменных по методике Чапина в исследуемой программе

Блок, переменные и директивы	Роль	Вход	Выход	Роль	Выход (цель)
S1: (инициализация 1) <Windows.h>; <iostream>; <conio.h>; <clocale>; <cstdio>; namespace std;	***T*** ***T*** ***T*** ***T*** ***T*** ***T***	Обеспечить доступ к Windows API (для ввода с клавиатуры и мыши). Управление языковым стандартом (локалью). Использовать стандартную	Обеспечить доступ к Windows API (для вывода на экран). Обеспечить вывод на экран.	***T*** ***T*** ***T*** ***T*** ***T*** ***T***	Вывод полученных значений.

hstdout	***T***	библиотеку языка. Дескриптор, обеспечить вывод на консоль.			***T***	
S2: (инициализация 2)						
$F1_{НАЧ\ MIN}$	***P***	Эти данные будут				
$F2_{НАЧ\ MIN}$	***P***	изменены в				
$F1_{НАЧ\ MAX}$	***P***	программе по		Нет влияния.		
$F2_{НАЧ\ MAX}$	***P***	значению в результате выполнения функции ПО, и станут её выходными данными.				.
n	***P***	Задаёт условие для принятия решения.		Нет влияния.		
b	***P***	Задают параметры		Нет влияния.		
dx	***P***	для работы в цикле.				
a	***P***	Задаёт значение для расчёта функций.		Нет влияния.		
x_0	***P***	Задаёт начальное значение аргумента для расчёта функций.		Нет влияния.		
S3: (задание вых. атрибутов 1)						
max2	***C***	Активирует вывод нижней строки таблицы, используются сведения из тела подчинённого цикла.		Вывод нижней части таблицы.	***T***	Обеспечить представление таблицей.
S5: (вывод результата)						
min1	***P***, ***T***	Данные для выводимых		Результат расчёта.	***M***	
max1	***P***, ***T***	результатов.		Результат расчёта.	***M***	Результаты
max2	***P***, ***T***			Результат расчёта.	***M***	расчёта.
min2	***P***, ***T***			Результат расчёта.	***M***	
S6: (счётчик циклов)						
i	***P***, ***C***	Начальное состояние счётчика циклов i=0. Условие для принятия решения i<n – это элемент выхода из итерации. Инкремент для поиска решения i++.		Указатель для поиска.	***T***	Обеспечить поиск результата и его запись в таблицу.

ПРИЛОЖЕНИЕ Г

Пример расчёта меры сложности Q исследуемой программы по методу Чапина

Пример расчёта значения сложности ***Q*** программы приведён частично, отражая методику Чапина.

1. Для каждого блока, используя таблицу В.1 ролей данных, определим количество элементов ***C***, ***P*** и ***T*** **для входа**, и ***M*** и ***T*** как элементов **выходных данных**. Если один элемент имеет несколько ролей, несколько источников или назначений, каждый из них будет учтён. Все данные сведём в таблицу Г.1, **расположив данные с ролями для входа и для выхода** программы **в колонках от второй до пятой**.

2. Для каждого блока общее количество каждой роли данных умножаем на свой весовой коэффициент W_i (на **3** для данных ***C***, на **2** для ***M***, на **1** для ***P*** и **0,5** для данных ***T***).

3. Просуммируем взвешенные значения элементов данных для каждого блока.

4. Принимаем значение числа процедур E_i, содержащих данные ***C***, определяющие выход из итераций циклов, **равным нулю**. Затем **определяем, какие блоки содержат данные для выхода из итераций в случае, когда подчинённые** (в том числе, вложенные) **блоки являются частью итеративно вызываемого тела цикла**. Для этого удобно использовать диаграмму древовидной структуры, что даст иерархию задач и подзадач для блоков (*в примере не приводится*). **Если циклы и их итерации выполняются в блоке при отсутствии подчинённых модулей или блоков**, участвующих в итерациях цикла, **то такие циклы и итерации игнорируются**.

5. Для каждого выходящего из итераций блока, определённого в п. 4, проверяем элементы ***C***, **чтобы определить, какие из них являются обслуживающими в определении выхода из итераций подчинённых блоков**, входящих в состав цикла. **Определяем, где возникают эти данные** роли ***C***. Если они **берутся только из выходящего из итераций блока или являются константами, добавляем ноль** к значению ***E*** при определении значения R_i. Так рассматриваются все элементы роли ***C***.

Если они **берутся из тела подчинённого цикла, добавляем «1» к значению *E*** при определении величины R_i,. Если они **берутся извне тела цикла, то добавляем «2» к значению *E*** при определении R_i. Если такой элемент роли ***C*** **изменяется ещё и внутри тела цикла, добавляем ещё «1» к значению *E***, чтобы получить «**3**».

6. Используем значение величины ***E*** каждого блока **для определения фактора повторения *R*** с помощью формулы (2) – **прибавляется к «1» квадрат одной трети от *E***. Значения *R* для значений *E* обычно следующие: *E*=0 даёт *R*=1,00; *E*=1 даёт *R*=1,11; *E*=2 даёт *R*=1,44; *E*=3 даёт *R*=2,00; ... .

7. Умножаем **сумму взвешенных данных** ΣW_i из п. 3 **на соответствующие значения** R_i блоков.

8. По формуле (1) **находим значение показателя** «**сложность**» Q_i отдельного *i*-го программного блока, используя значения произведений $\Sigma\ W_i \cdot R_i$, полученных в п. 7.

9. Определяем показатель сложности ***Q*** программы по формуле (4) вычислением среднего арифметического показателей сложности блоков – компонентов программы.

Таблица Г.1 – **Оценка сложности модулей программы-примера**

№ модуля	**Исходные данные**				**Взвешенные данные W_i**				**Сумма взвешенных данных ΣW_i**	**Фактор повторения R_i**	**$\Sigma\ W_i \cdot R_i$**	**Сложность модуля $Q_i=(\Sigma\ W_i \cdot R_i)^{0,5}$**
	C	***P***	***M***	***T***	***C***	***P***	***M***	***T***				
1	0	0	0	14	0	0	0	7	7 (*E*=0)	1,0	7,0	2,6
2	0	9	0	0	0	9	0	0	9 (*E*=0)	1,0	9,0	3
3	1	0	0	1	3	0	0	0,5	3,5 (*E*=1)	1,11	3,88	1,97
4	0	0	0	1	0	0	0	0,5	0,5 (*E*=0)	1,0	0,5	0,7
5	0	4	4	4	0	4	8	2	14 (*E*=0)	1,0	14	3,7
6	1	1	0	1	3	1	0	0,5	4,5 (*E*=1)	1,11	4,99	2,2
7	0	4	4	4	0	4	8	2	14 (*E*=0)	1,0	14,0	3,7
...	...	...	...	...	...	...	...	...	...	...	...	...
Общее число модулей *n*=12					Показатель сложности программы *Q*=3,01≈3,0. Сумма взвешенных данных ΣW_i=89.							

ПРИЛОЖЕНИЕ Д

Выводы по расчёту меры сложности исследуемой программы по методу Чапина

При оценке сложности программ параметр «**Сумма взвешенных данных ΣW_i**» отражает **сложность программы**, его **слагаемые позволяют судить о вкладе данных различной роли в сложность**.

Корректное представление ролей данных, а также **структура программы** и **особенности языка программирования** дают **объективное значение параметра ΣW_i**, что ведёт к адекватном суждению об **информационной прочности модуля, определяемой связностью элементов внутри модуля и сцеплением модуля с другими блоками**.

Также **рекомендуется отслеживать значение меры для последующих версий программ**, предполагая **появление ошибок при большом изменении меры Чапина для очередной версии**.

Метод Чапина информирует, что **значение показателя *Q* показывает, что будет представлять реальную сложность кода при реализации**: **чем больше данных используется, тем более сложной будет обработка данных**.

Нижняя граница сложности модуля, по метрике Чапина, **составляет 1,0**. Сложностью, равной нулю, **может обладать модуль, не имеющий данных с ролями *C*, *P*, *T*** или ***M***. Фрагменты в корневой части ПС обычно **несут функции обеспечения доступа к API** ОС, **управления языковым стандартом, использования стандартных библиотек языка и дескрипторов**, и т. п.

ПО, имеющее рациональную структуру, **реализованное по методике структурного программирования, имеет низкую сложность**.

Самая высокая сложность в структурированном ПО **приходится на основные ветви древовидной структуры**, но при этом **значение *Q* редко более 11. У листов дерева сложность** модулей **не превышает 5.**

Неструктурированное ПО имеет более высокую среднюю сложность.

Сложность программного модуля осуществляется **путём подсчета количества переменных каждой роли с учётом весовых коэффициентов, и значение *Q* отражает функциональную значимость переменных и их роль** в программы.

Анализ взаимодействия между частями программы позволяет оценить сложность управления потоками данных. Метрика Чапина – это **комплексный метод анализа, который учитывает и количество переменных, и их функциональное значение, а также влияние на потоки данных**.

Рассматривая **сложность *Q* программной системы из многих модулей, определяется среднее арифметическое показателей сложности модулей компонентов** ПС.

Показатель сложности **нужно определять с учётом распределения значений *Q* по модулям системы**, что п**оказывает сбалансированность распределения сложности по компонентам** системы.

При оценке сложности ПО, выполненного в парадигме объектно-ориентированного программирования, **классы нужно ассоциировать с модулями**. Модульность разделяет функциональность на независимые части. **Класс определяет структуру и поведение объекта, и позволяет создавать независимые компоненты программы**. **Каждый класс представляет отдельный модуль со своей логикой и данными**.

Когда ПО не является модульным, в исходном коде **в качестве модулей можно выделить лексические блоки, обладающие конкретным назначением, определить фрагменты, подпрограммы, процедуры, блоки**.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рыжков Е. Программный код и его метрики. – URL: http://habrahabr.ru/company/intel/blog/106082/ : 13.10.2010 (дата обращения 15.06.2020).

2. Решетникова Е.В. Метрология и качество программного обеспечения (Метрики, основанные на лексическом анализе программ): методические указания к организации аудиторной и самостоятельной работы студентов факультета информатики, математики и экономики, обучающихся по направлению подготовки 01.04.02 Прикладная математика и информатика, профиль «Математическое

моделирование» / Е.В. Решетникова; Новокузнецкий ин-т (фил.) Кемеров. гос. ун-та. – Новокузнецк : НФИ КемГУ, 2020 – 80 с. – https://skado.dissw.ru/indicationsvkr/2614/ (дата обращения 11.03.2021).

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