МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 Математическое обеспечение и администрирование информационных систем

(код и наименование направления подготовки, специальности)

Технология программирования

(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему «Разработка программного обеспечения для анализа экологических данных»

Студент	И.О. Назаров	
	(И.О. Фамилия)	(личная подпись)
Руководитель	О.В. Аникина	
	(И.О. Фамилия)	(личная подпись)
Консультанты	А.В. Прошина	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н, доцент, А.В. Очеповский _____

(степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 20___ г.

Тольятти 2018

Аннотация

Бакалаврская работа состоит из введения, трех глав, заключения, списка использованной литературы и приложений, и посвящена разработке приложения для анализа экологических данных.

Цель работы – разработать приложение для анализа экологических данных методом таблицы сопряжённости.

Объектом работы является процесс обработки экологических данных.

Предметом исследования в данной работе являются приложения, проводящие анализ экологических данных предприятий.

В первой главе рассматривается деятельность ООО «Региональный Экологический центр», его способ работы с экологическими данными, проводится сравнительный анализ функционала существующего программного обеспечения и формализация требований к новому.

Во второй главе рассматривается метод таблиц сопряжённости, способ его применения для предприятий, проектирование алгоритма работы программного обеспечения и его реализация.

В третьей главе проводится отладка и тестирование программного обеспечения.

В заключении делается вывод о проделанной работе. Разработанная программа высчитывает и выводит корректный результат и автоматизирует процесс расчётов оценки зависимости между экологическими данными предприятия, а также, выявление соответствия или несоответствия количества образовавшихся за год отходов предприятия норме.

Бакалаврская работа содержит пояснительную записку из 61 страниц, в том числе 19 рисунков, 5 таблиц, списка, 43 ссылок, в том числе 6 иностранных источников.

Abstract

This work consists of an introduction, three chapters, conclusion, bibliography and applications, and is devoted to the development of an application for environmental data analysis.

This thesis deals with the development of an application for the analysis of environmental data by the method of contingency tables.

The aim of the work is to develop an application for the analysis of environmental data by the method of the contingency table.

The object of the work is the method of conjugation table and its application in ecology.

The subject of the study in this paper is applications that conduct determinative data analysis.

The first Chapter discusses the activities of "Regional Environmental center", its way of working with environmental data, a comparative analysis of the functionality of existing software and formalization of requirements for the new.

In the second Chapter, the method of conjugation tables, the method of its application for enterprises, the design of the algorithm of the software and its implementation.

The third Chapter is testing and debugging software.

In conclusion, the conclusion is made about the work done. The developed program calculates and displays the correct result and automates the calculation process of assessing the relationship between the environmental data of the enterprise, as well as the identification of compliance or non-compliance of the amount of waste generated for the year to the norm.

The thesis consists of an explanatory note on page 61, introduction, including 19 figures, 5 tables, list, 43 references, including 6 foreign sources.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Важнейшим вопросом стратегии регулирования качества окружающей природной среды (ОПС) является вопрос создания системы, способной определять наиболее критические источники и факторы антропогенного воздействия на здоровье населения и ОПС, выделять наиболее уязвимые элементы и звенья биосферы, подверженные такому воздействию.

Такой системой признана система мониторинга антропогенных изменений состояния окружающей природной среды, способная представить необходимую информацию для принятия решений соответствующими службами, ведомствами, организациями.

Актуальность обуславливается тем, что в ООО «Региональный Экологический центр», занимаясь расчётом количества отходов и закупаемого сырья на, например, кондитерских предприятиях, специалисты выполняют математические операции с данными о сырье и отходах вручную. В связи с этим было решено разработать программное обеспечение для автоматизации процесса расчётов.

Объектом исследования является метод таблицы сопряжённости и его применение в экологии.

Предметом исследования в данной работе являются приложения для анализа экологических данных предприятий.

Практическая значимость данной работы обуславливается тем, что одним из видов деятельности предприятия ООО «Региональный Экологический центр» является расчёт образования отходов.

Для подсчёта результата на основе экологических данных был выбран метод таблиц сопряжённости, так как данный метод является наиболее удобным и простым для подсчёта корреляции на основе информации о количестве отходов.

Данная работа была представлена на конференциях:

- IV Международная научно-практическая конференция (школа-семинар) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук», которая пройдет в Тольяттинском государственном университете с 23 по 25 апреля 2018 года;
- Студенческие дни науки ТГУ 2018.

Целью данной работы является создание программного обеспечения, реализующего анализ экологических данных методом таблицы сопряжённости, основанных на информации о производственных отходах и сырье кондитерского предприятия, предоставленных ООО «Региональный Экологический центр».

Задачами данной работы являются:

- исследование деятельности ООО «Региональный Экологический центр»;
- проведение сравнительного анализа функционала существующего программного обеспечения;
- формализация требований для нового программного обеспечения;
- составление математической модели для разработки программного обеспечения;
- проектирование алгоритма работы нового программного обеспечения;
- реализация нового программного обеспечения;
- отладка и тестирование разработанного программного обеспечения.

Данная работа состоит из введения, трёх глав и заключения.

В первой главе рассматривается деятельность ООО «Региональный Экологический центр», проводится сравнительный анализ функционала существующего программного обеспечения и формализация требований к новому.

Во второй главе рассматривается метод таблиц сопряжённости, способ его применения для предприятий, проектирование и реализация самого программного обеспечения.

В третьей главе проводится отладка и тестирование приложения.

1 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Деятельность ООО «Региональный Экологический центр»

Проведем анализ деятельности предприятия ООО «Региональный Экологический центр». Один из видов деятельности данного предприятия – расчёт образования отходов на кондитерских предприятиях.

В результате производственной деятельности организации образуются промышленные отходы производства и потребления 1, 4, 5 классов опасности.

Классификация отходов по принадлежности, классу опасности, характеру действия приведена в таблице 1.1.

Таблица 1.1 – Классификация отходов по принадлежности, классу опасности, характеру действия

Признак классификации	**Наименование отходов**	**Количество, т/год**
Всего отходов		7,773
I. По классу опасности		
I класса опасности	Ртутные лампы, люминесцентные ртутьсодержащие трубки отработанные и брак	0,0002
Всего		0,0002
IV класса опасности	Мусор от бытовых помещений организаций несортированный (исключая крупногабаритный)	0,700
	Отходы производства пищевых продуктов	3,750
	Смет с территории	2,000
Всего		6,450
V класса опасности	Отходы упаковочного картона незагрязненные	1,250
	Отходы полипропилена в виде пленки	0,073
Всего		1,323
II. По характеру действия с отходами		
Размещаются на полигонах		
IV класса опасности	Мусор от бытовых помещений организаций несортированный (исключая крупногабаритный)	0,700
	Отходы производства пищевых продуктов	3,750
	Смет с территории	2,000
Всего		6,450
Итого		6,450
Передаются для обезвреживания, переработки, использования другими предприятиями		
I класса опасности	Ртутные лампы, люминесцентные ртутьсодержащие трубки отработанные и брак	0,0002

Признак классификации	Наименование отходов	Количество, т/год
Всего		0,0002
V класса опасности Всего	Отходы упаковочного картона незагрязненные	1,250
	Отходы полипропилена в виде пленки	0,073
Всего		1,323
Итого		1,3232

Основная деятельность предприятия – производство шоколада и сахаристых изделий (торты).

Производство кондитерских изделий состоит из трех основных стадий: приготовление крема, подготовка бисквитов и формирование тортов и пирожных.

В процессе формирования изделий образуются остатки кроя бисквитов и другие пищевые отходы. В результате деятельности предприятия образуются: отходы производства пищевых продуктов. Количество образования данного отхода составляет 15 кг/сут.

Сформированные кондитерские изделия упаковываются в пластиковые коробки и отправляются в магазины на реализацию на поддонах, являющихся оборотной тарой. Пластиковые контейнеры для кондитерских изделий поступают на комбинат в картонных коробках.

При растаривании сырья (масло, меланж, маргарин) и пластиковых коробок образуется отход: отходы упаковочного картона незагрязненные. Количество образования отхода составляет 1,250 т/год, отход поступает на переработку.

Мука и сахар поступают на предприятие в полипропиленовых мешках. При растаривании данного сырья образуется отход: отходы полипропилена в виде пленки. Мешки реализуют сотрудникам предприятия.

Соль, сухофрукты поступают в полиэтиленовой упаковке. В связи с малым количеством образования полиэтиленовой упаковки (менее 30 кг/год), данный отход учтен в мусоре от бытовых помещений организаций несортированный (исключая крупногабаритный).

В результате жизнедеятельности сотрудников образуется мусор от бытовых помещений организаций несортированный (исключая крупногабаритный). Канцелярская деятельность на предприятии не ведется.

При уборке прилегающей территории образуется отход: смет с территории.

В результате освещения помещений образуются ртутные лампы и люминесцентные ртутьсодержащие трубки, отработанные и брак. Отход временно, до отправки на утилизацию, складируются в специально отведенном месте в заводской упаковке в складском помещении на каждой площадке. В случае боя ламп осколки собираются в емкость из материала не амальгирующего и не адсорбирующего ртуть (стекло, винипласт) с плотно закрывающейся крышкой, заполненную раствором перманганата калия.

Таблица 1.2 – Перечень отходов, для которых устанавливается годовой норматив образования

Отходообразующий вид деятельности, процесс	**Наименование вида отхода**	**Код отхода по ФККО**	**Класс опасности**
Освещение помещений территории	Ртутные лампы и люминесцентные ртутьсодержащие трубки	353 301 00 13 01 1	1
Жизнедеятельность сотрудников предприятия	Мусор от бытовых помещений организаций несортированный (исключая крупногабаритный)	912 004 00 01 00 4	4
Производство пищевых продуктов (кондитерских изделий)	Отходы производства пищевых продуктов	111 000 00 00 00 0	4
Уборка территории	Смет с территории	912 000 00 00 00 0	4
Растаривание сырья и материалов	Отходы упаковочного картона незагрязненные	187 102 02 01 00 5	5
Растаривание сырья	Отходы полипропилена в виде пленки	571 030 02 01 99 5	5

Таблица 1.3 представляет состав и физико-химические свойства отходов.

Таблица 1.3 – Состав и физико-химические свойства отходов

Наименование вида отхода	**Отходообразующий вид деятельности, процесс**	**Класс опасности для окружающей среды**	**Опасные свойства**	**Физико-химические свойства отхода**		
				Агрегатное состояние	Наименование компонентов	Содержание компонентов, %
Ртутные лампы и люминесцентные ртутьсодержащие трубки, отработанные и брак	Освещение помещений, территории	1	Токсичность	ГИП ПС	Стекло Другие металлы Ртуть Прочее	92,00 2,00 0,02 5,98
Мусор от бытовых помещений организаций	Жизнедеятельность сотрудников предприятия	4	Данные не установлены	Твердый	Продукты растительного происхождения	4,14 78,30
Отходы производства пищевых продуктов	Производство пищевых продуктов (кондитерских изделий)	4	Данные не установлены	Твердый	Продукты природного растительного и животного происхождения Вода Целлюлоза Полиэтилен	62,23 34,19 2,12 1,46
Смет с территории	Уборка территории	4	Данные не установлены	Твердый	Грунт Вода Целлюлоза Нефтепродукты	89,65 6,91 3,41 0,03
Отходы упаковочного картона незагрязненные	Растаривание сырья и материалов	5	Данные не установлены	Твердый	Целлюлоза	100,00

Рассмотрим виды отходов кондитерского предприятия. Первый вид отходов – ртутные лампы и люминесцентные ртутьсодержащие трубки, отработанные и брак.

По данным предприятия, для освещения установлены лампы марок ЛБ-40, количество точек установки ламп 2 шт.; расчет норматива образования ртутных ламп, подлежащих утилизации ($К_л$), был произведен по формуле 1.1:

$$К_л = (К_{рл} \times Ч_р \times С) / Н_р , \quad (1.1)$$

где $К_{рл}$ – количество установленных ртутных ламп, шт.; $Ч_р$ – среднее время работы в сутки одной ртутной лампы (10,34 часа – для ламп уличного освещения, 4,57 часа – для ламп освещения помещений); С – число рабочих дней в году; $Н_р$ – нормативный срок службы одной лампы, час, а для ЛБ-40 – 12000 час.

Рассчитаем годовое количество отхода люминесцентных ламп:

$К_{л1}$ = 2* 4,57 * 250 /12000 = 1 шт./год.

Учитывая массу одной лампы, которая для ЛБ – 40 равна 210 г, рассчитаем массу отхода люминесцентных ламп подлежащих утилизации:

М= (1*0,21)/1000 = 0,0002 т/год.

Отход временно, до отправки на утилизацию, складируется в специально отведенном месте в заводской упаковке в складском помещении.

Второй вид отходов – мусор от бытовых помещений организаций несортированный (исключая крупногабаритный).

Количество сотрудников на предприятии, согласно штатному расписанию, составляет 10 человек. Количество данного отхода, образующихся на одного работника в год составляет 70 кг, при средней плотности отходов 200 кг/м^3. Годовое количество мусора от бытовых помещений организаций несортированный (исключая крупногабаритный) составит:

М = 10* 70/1000 = 0,700 т/год.

Третий вид – отходы производства пищевых продуктов.

Норматив образования отхода принят согласно данным предприятия и составляет 15 кг в сутки. Количество образования отхода составит:

М = 250*15/1000 = 3,750 т/год.

Четвёртый вид – смет с территории.

Расчет предельного образования отхода был произведен исходя из норм образования смета на усовершенствованных покрытиях (асфальт, бетон - 10 кг/м^2 в год) [10]. Площадь убираемой территории 200 м^2.

Следовательно, предельное количество отхода составит:

М = 200 * 10/1000 = 2,000 т/год.

Пятый вид – отходы упаковочного картона незагрязненные.

При растаривании сырья (масло, меланж, маргарин) и пластиковых коробок образуется данный отход. Количество образования отхода, по данным предприятия, составляет 1,250 т/год.

Шестой вид – отходы полипропилена в виде пленки.

Мука и сахар поступают на предприятие в полипропиленовых мешках по 50 кг. При растаривании сырья образуется данный отход.

Таблица 1.4 – Количество закупаемого сырья, мешков, вес загрязнённого мешка и количество образования отхода

Сырье	Количество закупаемого сырья, т/год	Количество закупаемых мешков, шт./год	Вес одного загрязненного мешка, кг	Количество образования отхода, т/год
Мука	24,000	480	0,090	0,043
Сахар	16,800	336	0,090	0,030
				0,073

Используя данные из таблицы 1.4 можно вычислить оценку зависимости количества образования отхода от количества закупаемого сырья и выяснить, соответствует ли количество отходов норме или нет. Данные вычисления позволяет сделать анализ методом таблицы сопряжённости.

1.2 Анализ существующего программного обеспечения для обработки экологических данных

Рассмотрим существующие программные продукты, предназначенные для работы с экологическими данными предприятий, и проведем сравнительный

анализ наличия или отсутствия в них того или иного функционала для работы с этими данными [1].

Первая программа, которую мы рассмотрим – «2-ТП – отходы (обзор)». Она предназначена для автоматического приема, обработки и хранения данных, содержащихся в форме статистической отчетности «2-ТП Отходы» и используется в территориальных органах охраны окружающей среды.

Данная программа позволяет принимать отчеты предприятий по форме 2-ТП (отходы) как в режиме ручного ввода, так и с магнитного носителя в виде символьного файла специального формата.

Также в программе возможно ведение единого кодификатора отходов, баз данных природопользователей, объектов размещения отходов.

Для подготовки отчета по форме 2-ТП (отходы) можно использовать программу «2-ТП (отходы)». Программа «2-ТП (отходы)» позволяет подготовить данные отчета в виде символьного файла, формат которого открыт и может быть подготовлен любой другой программой (в том числе текстовым редактором). Символьный файл может быть передан в территориальные органы на любом магнитном носителе [2].

На основании принятых отчетов предприятий формируются следующие таблицы для обзора на территории республики (края, области, автономного округа):

- отчет о природопользователях;
- отчет по объектам размещения (захоронения) отходов;
- сводные таблицы государственной статистической отчетности по форме 2-ТП (токсичные отходы) в разрезе области, городов, районов, природопользователей, полигонов и свалок.

Программный комплекс «2-ТП (Обзор) – Отходы» может быть доработан с учетом региональных требований к организации обращения с отходами и таблицам статистической отчетности.

Программа предназначена для работы в системе Windows 95/98/Me/NT/2000/XP, что является одним из её недостатков.

Далее рассмотрим ЛИС «Химик-аналитик» для внутрилабораторного контроля» (ВЛК). Это специализированная версия ЛИС «Химик-аналитик». Она предназначена для организации ведения оперативного контроля и контроля стабильности результатов анализа в лаборатории. Кроме того, сохраняется возможность ведения вспомогательных журналов ЛИС «Химик-аналитик».

Рассмотрим комплектацию данной программы [4].

ЛИС «Химик-аналитик» для ВЛК – готовый «коробочный» программный продукт, который включает:

- компакт-диск с дистрибутивом «ЛИС «Химик-аналитик» для внутрилабораторного контроля»;
- руководство администратора;
- руководство пользователя.

Назначением программы является:

- ввод и хранение исходной информации о предприятии, его подразделениях, лабораториях, технологических установках, контрольных точках, контролируемых объектах анализа, используемых методиках анализа, алгоритмах контроля;
- ведение, для целей внутрилабораторного контроля, электронных лабораторных журналов с проверкой приемлемости результатов определений контролируемых параметров рабочих проб по ГОСТ Р ИСО 5725 или с контролем повторяемости результатов контрольных определений по РМГ 76;
- организация оперативного контроля процедур анализа по РМГ 76;
- организация контроля стабильности результатов анализа по ГОСТ Р ИСО 5725 и РМГ 76;
- установление показателей качества результатов измерений при реализации методик анализа в лаборатории по РМГ 76;

- автоматизированный документооборот аналитической лаборатории для целей внутрилабораторного контроля;
- проверка качества реактивов с просроченным сроком хранения по РМГ 59 и ПНД Ф 12.10.1.
- расчет градуировочных характеристик по ГОСТ Р ИСО 11095; РМГ 54 и МУ 6/113-30-19, а также контроль стабильности градуировочных зависимостей.

Дополнительные возможности данной программы [3]:

- ведение вспомогательных журналов учета. То есть, ведение учёта прихода и расхода реактивов, стандартных образцов, стандарт-титров; приготовления растворов, в том числе расчет метрологических характеристик аттестованных смесей по РМГ 60;
- организация системы менеджмента качества лаборатории по ГОСТ ИСО/МЭК 17025;
- межлабораторная метрологическая аттестация стандартных образцов по ГОСТ 8.532;
- межлабораторные сличения по РМГ 58.

Далее рассмотрим программный комплекс «Отходы-объединение», который предназначен:

- для контроля за образованием отходов, их получением, передачей или обезвреживанием. А также, размещением на предприятиях объединения, вывозом на свалки или захоронением;
- для формирования банка данных инвентаризации отходов и мест их размещения;
- для ведения статистической отчетности по форме 2-ТП (отходы);
- для учета полученных разрешений на размещение отходов;
- для контроля (расчета) платежей за размещение отходов.

Программный комплекс «Отходы – объединение» формирует комплект обязательной выходной документации в соответствии с существующими

требованиями к ее оформлению, а также различные сводные перечни и таблицы.

Комплекс состоит из трех модулей [5]:

- инвентаризация отходов и мест их размещения (ведения унифицированного перечня (каталога) отходов, инвентаризации мест размещения отходов, учета лимитов и разрешений на размещение отходов, полученных предприятиями и так далее);
- учет обращения с отходами (анализа информации по объединению об образовании, получении, передаче и размещении отходов с возможностью ввода оперативных данных в момент возникновения события, связанного с какой-либо операцией по обращению с отходами);
- справочники (справочник отходов, каталог-классификатор отходов, справочник компонентов отходов).

Далее рассмотрим программный комплекс «Отходы-объединение», который Предназначен [6]:

- для контроля за образованием отходов, их получением, передачей или обезвреживанием. А также, размещением на предприятиях объединения, вывозом на свалки или захоронением;
- для формирования банка данных инвентаризации отходов и мест их размещения;
- для ведения статистической отчетности по форме 2-ТП (отходы);
- для учета полученных разрешений на размещение отходов;
- для контроля (расчета) платежей за размещение отходов.

Программный комплекс «Отходы – объединение» формирует комплект обязательной выходной документации в соответствии с существующими требованиями к ее оформлению, а также различные сводные перечни и таблицы.

Комплекс состоит из трех модулей:

- инвентаризация отходов и мест их размещения (ведения унифицированного перечня (каталога) отходов, инвентаризации мест размещения);
- отходов, учета лимитов и разрешений на размещение отходов, полученных предприятиями и так далее);
- учет обращения с отходами (анализа информации по объединению об образовании, получении, передаче и размещении отходов с возможностью ввода оперативных данных в момент возникновения события, связанного с какой-либо операцией по обращению с отходами);
- справочники (справочник отходов, каталог-классификатор отходов, справочник компонентов отходов).

В таблице 1.6 наглядно представлено наличие или отсутствие функционала рассмотренных выше программ для анализа экологических данных [7].

Таблица 1.6 – Наличие и отсутствие в рассмотренных программах функционала для анализа экологических данных

Функционал	«2-ТП – отходы (обзор)»	ЛИС «Химик-аналитик»	«Отходы – объединение»
Создание отчётов	+	+	+
Расчёт количества отходов	–	–	–
Хранение экологических данных и отчётов о них	+	+	+
Расчёт зависимости одних экологических данных от других	–	–	–
Сравнение количества получившихся отходов с их нормой	–	–	–

Из таблицы 1.6 очевидно, что ни одна из рассмотренных программ не рассчитывает, сильно ли зависит количество отходов от количества закупаемого сырья и, стоит отметить, являются довольно дорогими и громоздкими. Поэтому было решено разработать программное обеспечение, которое поможет ответить на данный вопрос [9].

1.3 Формализация требований к новому программному обеспечению

Сформулируем формальные требования к разработанному программному обеспечению.

Программа должна:

- предложить ввести пользователю строк и столбцов таблицы сопряжённости в диалоговых окнах;
- создать на основе введённых данных пустую таблицу;
- рассчитать результат на основе введённых в таблицу данных.

На рисунке 1.1 представлена диаграмма прецедентов разрабатываемой программы.

Рисунок 1.1 – Диаграмма прецедентов нового программного обеспечения

Данная диаграмма наглядно отражает формализацию требований к новому программному обеспечению [10]. В таблице 1.6 представлена спецификация прецедента «Расчёт χ^2, коэффициента взаимной сопряжённости Пирсона и проверка соответствия отходов норме».

Таблица 1.7 – спецификация прецедента «Расчёт χ^2, коэффициента взаимной сопряжённости Пирсона и проверка соответствия отходов норме»

Прецедент: Расчёт χ^2, коэффициента взаимной сопряжённости Пирсона и проверка соответствия отходов норме
ID: 1
Краткое описание: Программа рассчитывает χ^2 и коэффициенты взаимной сопряжённости таблицы сопряжённости
Главные актёры: Пользователь

Второстепенные актёры: Нет.
Предусловия: 1. Пользователем должны быть введены количества строк и столбцов таблицы сопряжённости. 2. Пользователем должны быть введены названия строк и столбцов таблицы сопряжённости. 3. Пользователем должна быть введена годовая норма отходов. 4. Пользователем должны быть введены данные в таблицу сопряжённости.
Основной поток: 1. Прецедент начинается, когда Пользователь ввёл корректные, то есть, не содержащие букв данные в таблицу сопряжённости, и нажал «Enter».
Постусловия: Нет.
Альтернативные потоки: Нет.

Далее в таблице 1.8 представлена спецификация прецедента «Проверка введённых в таблицу сопряжённости данных».

Таблица 1.8 – спецификация прецедента «Проверка введённых в таблицу сопряжённости данных»

Прецедент: Проверка введённых в таблицу сопряжённости данных
ID: 2
Краткое описание: Программа проверяет, введённые в таблицу сопряжённости Пользователем, данные.
Главные актёры: Нет.
Второстепенные актёры: Нет.

Предусловия: 1. Пользователем должны быть введены количества строк и столбцов таблицы сопряжённости. 2. Пользователем должны быть введены названия строк и столбцов таблицы сопряжённости. 3. Пользователем должна быть введена годовая норма отходов. 4. Пользователем должны быть введены данные в таблицу сопряжённости.
Основной поток: 1. Прецедент начинается, когда Пользователь нажал на «Enter», для начала проверки и дальнейших расчётов на основе данных, введённых в таблицу сопряжённости.
Постусловия: Нет.
Альтернативные потоки: Нет.

Ниже в таблице 1.9 представлена спецификация прецедента «Сообщение о введённых в таблицу сопряжённости буквах».

Таблица 1.9 – спецификация прецедента «Сообщение о введённых в таблицу сопряжённости буквах»

Прецедент: Сообщение о введённых в таблицу сопряжённости буквах
ID: 3
Краткое описание: Программа выводит на экран окно с сообщением, если Пользователь ввёл хотя бы одну букву в таблицу сопряжённости.
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Пользователем должны быть введены количества строк и столбцов таблицы сопряжённости. 2. Пользователем должны быть введены названия строк и столбцов таблицы сопряжённости. 3. Пользователем должна быть введена годовая норма отходов. 4. Пользователем должны быть введены данные в таблицу сопряжённости. 5. Программа начала расчёт, на основе данных из таблицы сопряжённости.

Основной поток: 1. Прецедент начинается, если Пользователь ввёл в таблицу сопряжённости хотя бы одну букву и нажал на «Enter» для начала расчётов.
Постусловия: Нет.
Альтернативные потоки: Нет.

По данной главе можно сделать следующий вывод: так как, ни одно из рассмотренных программных обеспечений не выполняет расчёт зависимости между экологическими данными, например, зависимости количества отходов, вырабатываемых предприятием от количества закупаемого сырья, или проверка соответствия суммы отходов, произведённых за год, норме. Данные расчёты специалистам приходится проделывать вручную.

Поэтому было решено разработать программное обеспечение, помогающее рассчитать зависимость между экологическими данными.

1.4 Постановка задачи

В данной работе разрабатывается программное обеспечение для анализа экологических данных кондитерского предприятия. Специалисты, которые работают в ООО «Региональный Экологический центр», проводят расчёты с экологическими данными кондитерского предприятия, например, сравнивают сумму отходов предприятия с их годовой нормой или вычисляют, сильно ли зависит количество отходов от количества закупаемого сырья. Однако, они делают данные расчёты вручную, поэтому было решено разработать программное обеспечение, автоматизирующее данный процесс.

Для достижения цели, а именно, создания данного программного обеспечения, был составлен алгоритм на основе метода таблицы сопряжённости, который помогает рассчитать силу зависимости между экологическими данными (например, зависимость между данными о купленной муке, в т/год, и отходов,

полученных после её использования в производстве, также т/год). Стоит отметить, что в программное обеспечение было также добавлено сравнение суммы отходов с их годовой нормой.

Входные данные в программе:

- вводятся количество строк и столбцов;
- затем, годовая норма отходов (т/год);
- экологические данные (виды сырья, т/год и отходы от них, т/год).

Программа выводит:

- оценку силы связи $\chi^2_{\text{табл;}}$
- коэффициент взаимной сопряжённости Пирсона;
- результат сравнения суммы отходов с их годовой нормой (в виде текста «Сумма отходов в пределах нормы» или «Сумма отходов превышает норму»).

Также стоит заметить, что $\chi^2_{\text{табл}}$ сравнивается с $\chi^2_{\text{факт}}$, которую высчитывает специалист. Если $\chi^2_{\text{факт}} > \chi^2_{\text{табл}}$, то делается вывод, что элементы таблицы зависят друг от друга, а для более точного вывода о зависимости, высчитывается коэффициент взаимной сопряжённости Пирсона и чем он ближе к единице, тем сильнее зависимость.

2 РАЗРАБОТКА НОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ АНАЛИЗА ЭКОЛОГИЧЕСКИХ ДАННЫХ

2.1 Математическая модель разрабатываемого программного обеспечения

Таблица сопряжённости является наиболее универсальным средством изучения статистических связей (в том числе и между экологическими данными), так как в ней могут быть представлены переменные с любым уровнем измерения [11].

Строки таблицы сопряжённости соответствуют значениям одной переменной, столбцы — значениям другой переменной, при этом количественные шкалы предварительно должны быть сгруппированы в интервалы. Например, таблица сопряжённости может быть использована для показа зависимости количества отходов от количества закупаемого сырья [12].

Одной из задач статистики является анализ связи двух качественных признаков, каждый из которых представлен в виде альтернативных признаков. По характеру распределения можно судить случайно оно или нет, т.е. есть ли зависимость между изучаемыми признаками или нет. Анализ взаимосвязи между атрибутивными (качественными) признаками проводится на основе таблиц взаимной сопряженности, которые описывают комбинационные распределения совокупностей по двум признакам – факторному x и результативному y.

Оценка силы связи проведена при помощи критерия χ^2 (формула 1.2). С помощью данного критерия проверяется следующая гипотеза - гипотеза H0: переменные x и y независимы. Пусть имеется таблица сопряженности с k_1 строк и k_2 столбцов, построенная для переменных x и y [14].

$$\chi^2 = \mathbf{n} \quad \frac{n_{xy}^2}{n_x n_y} - 1 \qquad (1.2)$$

где n_x, n_y – суммы частот в соответствующей строке и столбце, n_{xy} – частота в конкретной ячейке, n – сумма значений всех ячеек [11].

Чтобы понять, каким является распределение данных о количестве отходов и сырья: зависят ли друг от друга элементы таблицы или нет, высчитывается табличное значение χ^2 для выбранного уровня значимости α. Число степеней свободы тоже выбирается. В формуле 1 k_1, k_2 - число групп по строкам и столбцам в таблице сопряженности, далее если $\chi^2_{факт} > \chi^2_{табл}$, то делается вывод, что элементы таблицы зависят друг от друга и можно говорить о зависимости между признаками [22].

Отсюда следует, что χ^2 помогает выяснить, есть ли зависимость между элементами таблицы [23]. В новой программе получены результаты расчёта критерия $\chi^2_{табл}$, сравнивая который с $\chi^2_{факт}$ определяется зависимость количества отходов от количества сырья.

Для измерения тесноты связи используют коэффициенты взаимной сопряженности. В своей работе мы рассматриваем три из них: коэффициент взаимной сопряженности Крамера, Пирсона и Чупрова [13]. Коэффициент Крамера рассчитывается по формуле

$$C = \sqrt{\frac{\chi^2}{n\ \overline{(k_{min}-1)}}}, \qquad (1.3)$$

(если $k_1 \neq k_2$, где k_1, k_2 - число строк и столбцов в исходной матрице), k_{min} – минимальное значение, которое выбирается между k_1 и k_2, n - общее количество наблюдений, Пирсона – по формуле

$$G = \sqrt{\frac{\chi^2}{\chi^2+1}}, \qquad (1.4)$$

Чупрова – по формуле

$$C = \sqrt{\frac{\chi^2}{n\ \overline{(k_1-1)(k_2-1)}}} \qquad (1.5)$$

В новой программе лучше всего рассчитывать коэффициент Пирсона, так как его формула достаточно проста и более надёжна для её программной реализации. Коэффициент Пирсона рассчитывается наряду с $\chi^2_{табл}$, чтобы более

точно измерить тесноту зависимости экологических данных друг от друга (чем коэффициент Пирсона ближе к единице, тем сильнее зависимость) [25].

2.2 Общая структура алгоритма решения задачи

Математическая модель нового программного обеспечения рассмотрена, поэтому приступим к разработке основного алгоритма расчёта программы [15].

На рисунке 2.1 показана схема общего алгоритма работы программы.

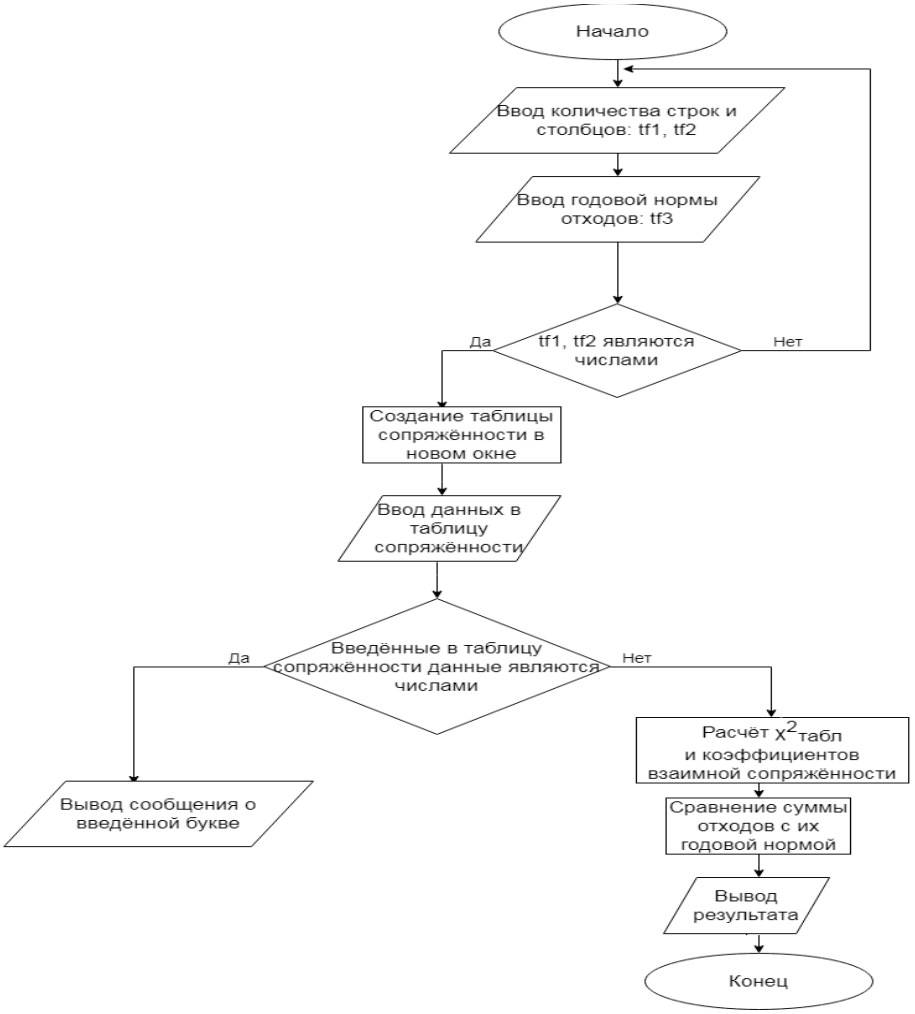

Рисунок 2.1 – Схема общего алгоритма работы программы.

Данный алгоритм описывает действия программы в общем. Программа предлагает пользователю ввести количество строк и столбцов в диалоговых окнах. Затем, на основе введённых пользователем чисел, создаётся таблица сопряжённости с пустыми ячейками. Далее пользователь вводит в ячейки таблицы сопряжённости данные и нажимает «Enter» для проверки введённых в таблицу

данных на наличие или отсутствие введённых в ячейки букв. Если буквы отсутствуют в ячейках таблицы сопряжённости, то программное обеспечение рассчитывает $\chi^2_{табл}$ и коэффициенты взаимной сопряжённости и выводит результат под таблицей [17].

Теперь рассмотрим поподробнее алгоритм расчётов программным обеспечением $\chi^2_{табл}$ и коэффициентов взаимной сопряжённости. Ниже на рисунке 2.2 продемонстрирован общий алгоритм расчётов, проводимых новым программным обеспечением.

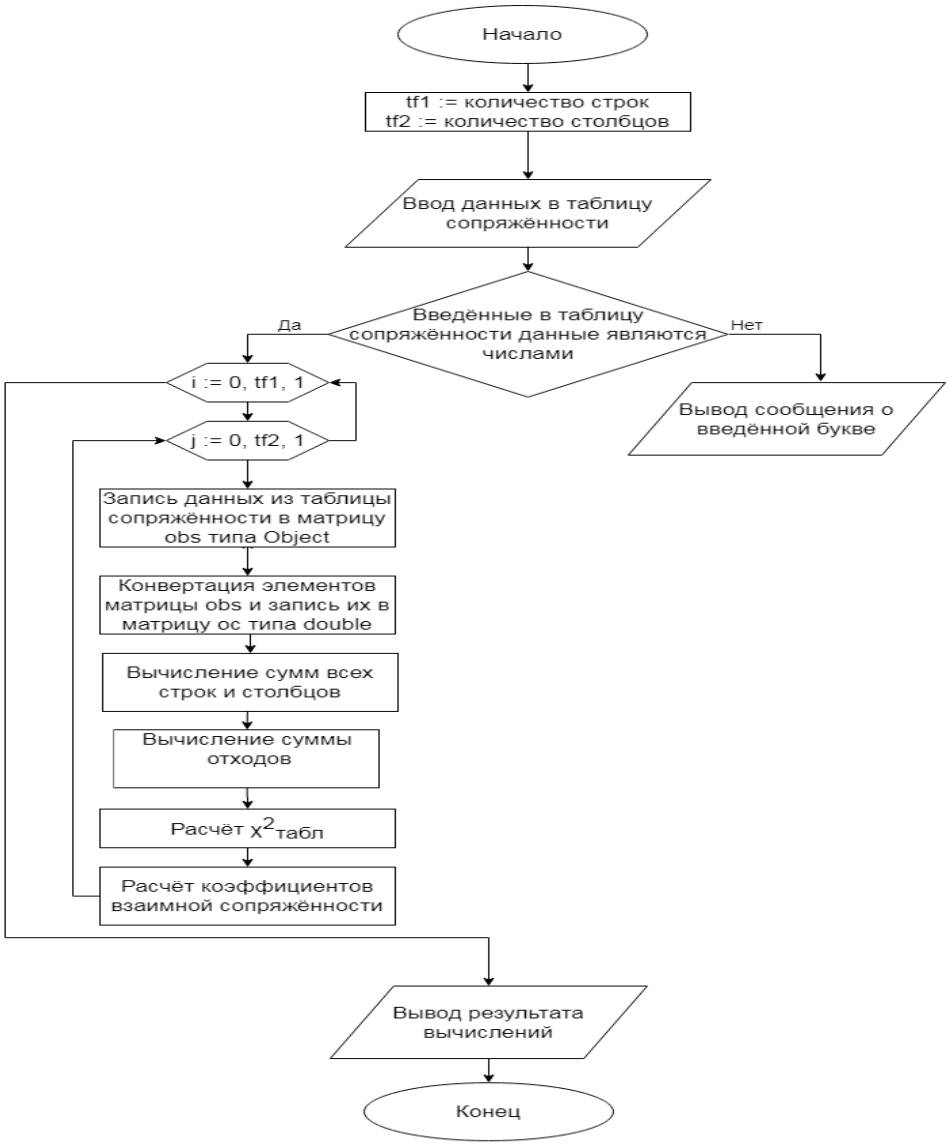

Рисунок 2.2 – Алгоритм расчётов, проводимых разрабатываемой программой

В данном алгоритме более подробно расписан ход действий программного обеспечения с введёнными в таблицу сопряжённости данными. Сначала программа проводит проверку данных, введённых в ячейки таблицы сопряжённости пользователем. Для этого данные, которые введены в ячейки, записываются в матрицу obs типа Object. Затем, данная матрица проверяется на наличие в ней букв, так как, в ячейках таблицы сопряжённости допускаются только цифры. Если данные введены корректно, то есть, среди них нет ни одной буквы, то матрица obs конвертируется в матрицу oc типа double. В матрице oc высчитываются суммы строк и столбцов записываются в одномерные массивы nx и ny соответственно. Далее, начинаются расчёты по формулам (1.2) – (1.5) и выводится результат под таблицей сопряжённости [16].

В объектно-ориентированном программировании между классами, являющимися, в совокупности, реализацией программного обеспечения могут быть различные связи. Чтобы понять, какая связь должна быть между классами нового программного обеспечения, рассмотрим суть этих связей (или другими словами, отношений между классами).

Рассмотрим суть зависимости. Суть её заключается в том, что это семантическая связь между зависимыми и независимыми элементами модели. Она существует между двумя элементами, если изменения в определении одного элемента (сервер или цель) могут вызвать изменения в другом (клиент или источник). Эта связь является однонаправленной. Зависимость является более слабой формой связи, которая указывает, что один класс зависит от другого, потому что он использует его в определенный момент времени. Один класс зависит от другого, если независимый класс является переменной параметра или локальной переменной метода зависимого класса. Это отличается от ассоциации, где атрибут зависимого класса является экземпляром независимого класса. Иногда отношения между двумя классами очень слабые. Они вообще не реализуются с

переменными-членами. Скорее они могут быть реализованы как аргументы функции-члена.

Суть ассоциации – семейство связей. Двоичная ассоциация (с двумя концами) обычно представлена в виде строки. Ассоциация может связывать любое количество классов. Ассоциация с тремя связями называется тройной ассоциацией. Ассоциация может быть названа, а ее концы могут быть украшены именами ролей, показателями владения, кратностью, видимостью и другими свойствами. Существует четыре различных типа ассоциаций: двунаправленная, однонаправленная, агрегация (включает агрегацию состава) и рефлексивная. Двунаправленные и однонаправленные ассоциации являются наиболее распространенными. Ассоциация представляет собой статическое отношение между объектами двух классов [18].

Суть агрегации – вариант отношения ассоциации «has»; агрегация более специфична, чем ассоциация. Это ассоциация, представляющая часть-целое или часть-отношения. Как тип ассоциации, агрегат может быть назван и иметь тот же вид, что и ассоциация. Однако агрегация может включать не более двух классов; она должна быть двоичной ассоциацией. Кроме того, вряд ли существует разница между агрегатами и ассоциациями во время реализации, и диаграмма может полностью пропустить отношения агрегации. Агрегирование может происходить, когда класс является коллекцией или контейнером других классов, но содержащиеся в нем классы не имеют сильной зависимости жизненного цикла от контейнера. Содержимое контейнера по-прежнему существует при уничтожении контейнера [20]. В UML агрегация графически представлена в виде полого ромба на содержащем классе с одной линией, которая соединяет его с содержащимся классом. Агрегат является семантически расширенным объектом, который рассматривается как единица во многих операциях, хотя физически он состоит из нескольких меньших объектов [21].

UML-представление отношения композиции показывает композицию как заполненную ромбовидную фигуру на содержащем конце класса линий, которые соединяют содержащиеся классы с содержащим классом [19]. Таким образом, связь агрегации часто является «каталогизированной» локализацией, чтобы отличить ее от «физической» локализации композиции.

Наследование (или обобщение), по сути, указывает на то, что один из двух связанных классов считается специализированной формой другого, а суперкласс считается обобщением подкласса. На практике означает, что любой экземпляр подтипа также является экземпляром суперкласса. Связь легче всего понять по фразе «А-Б» (мука-сырьё, сырьё-ресурс). Графическое представление обобщения в UML представляет собой полую треугольную фигуру на конце суперкласса линии (или дерева линий), которая соединяет ее с одним или несколькими подтипами. Отношение обобщения также называется отношением наследования или «is» [23]. Суперкласс (базовый класс) в отношении обобщения также известен как «родительский», суперкласс, базовый класс или базовый тип. Подтип в отношении специализации также известен как «дочерний», подкласс, производный класс, производный тип, наследующий класс или наследующий тип. Стоит обратить внимание, что эти отношения не имеют никакого сходства с биологическими отношениями родитель-ребенок: использование этих терминов чрезвычайно распространено, но может вводить в заблуждение [24].

Наследование может быть показано только на диаграммах классов и диаграммах вариантов использования.

Отношение реализации, по сути, является отношением между двумя элементами модели, в котором один элемент модели (клиент) реализует (реализует или выполняет) поведение, заданное другим элементом модели (поставщиком) [25]. Графическое представление реализации в UML представляет собой полую треугольную фигуру на интерфейсном конце пунктирной линии (или дерева линий), которая соединяет ее с одним или несколькими исполнителями [8].

Простая головка стрелки использована на конце интерфейса штриховой линии которая соединяет ее к своим потребителям. Реализации могут быть показаны только на диаграммах классов или компонентов. Суть реализации, другими словами, это связь между классами, интерфейсами, компонентами и пакетами, которая соединяет клиентский элемент с элементом поставщика. Отношение реализации между классами или компонентами и интерфейсами показывает, что класс или компонент реализует операции, предлагаемые интерфейсом [26].

Множественность связей указывает, что по крайней мере один из двух связанных классов ссылается на другой. Это отношение обычно описывается как «А имеет В».

Таблица 2.1 показывает все варианты количества экземпляров объектов.

Таблица 2.1 - все варианты количества экземпляров объектов

0	Нет экземпляров (редко)
0..1	Нет экземпляров или один экземпляр
1	Ровно один экземпляр
1..1	Ровно один экземпляр
0..*	Нуль или более экземпляров
*	Нуль или более экземпляров
1..*	Один или несколько экземпляров

Далее, в таблице 2.2 наглядно показан проделанный анализ сути каждой из зависимостей между классами, описанный выше.

Таблица 2.2 – краткая суть зависимостей между классами

Связь между классами	**Суть**
Зависимость	Она существует между двумя элементами, если изменения в определении одного элемента могут вызвать изменения в другом.
Ассоциация	Статическое отношение между объектами двух классов (классы содержат лишь ссылку на объект другого класса)
Агрегация	Ответвление ассоциации, представляющая часть-целое или часть-отношения.
Реализация	Отношение между двумя элементами модели, в котором один элемент модели реализует (реализует или выполняет)

Связь между классами	Суть
	поведение, заданное другим элементом модели.
Множественность	По крайней мере один из двух связанных классов ссылается на другой

Из проанализированных выше зависимостей, стало понятно, что для разрабатываемого программного обеспечения наиболее подходящей является ассоциация, так как класс-родитель должен содержать в себе лишь ссылку на объект дочернего класса, что, также хорошо показано в таблице 2.2. А, исходя из таблицы 2.1, был сделан вывод, что связь между классами будет один к одному.

Далее, на рисунке 2.3, представлена диаграмма классов разрабатываемого программного обеспечения.

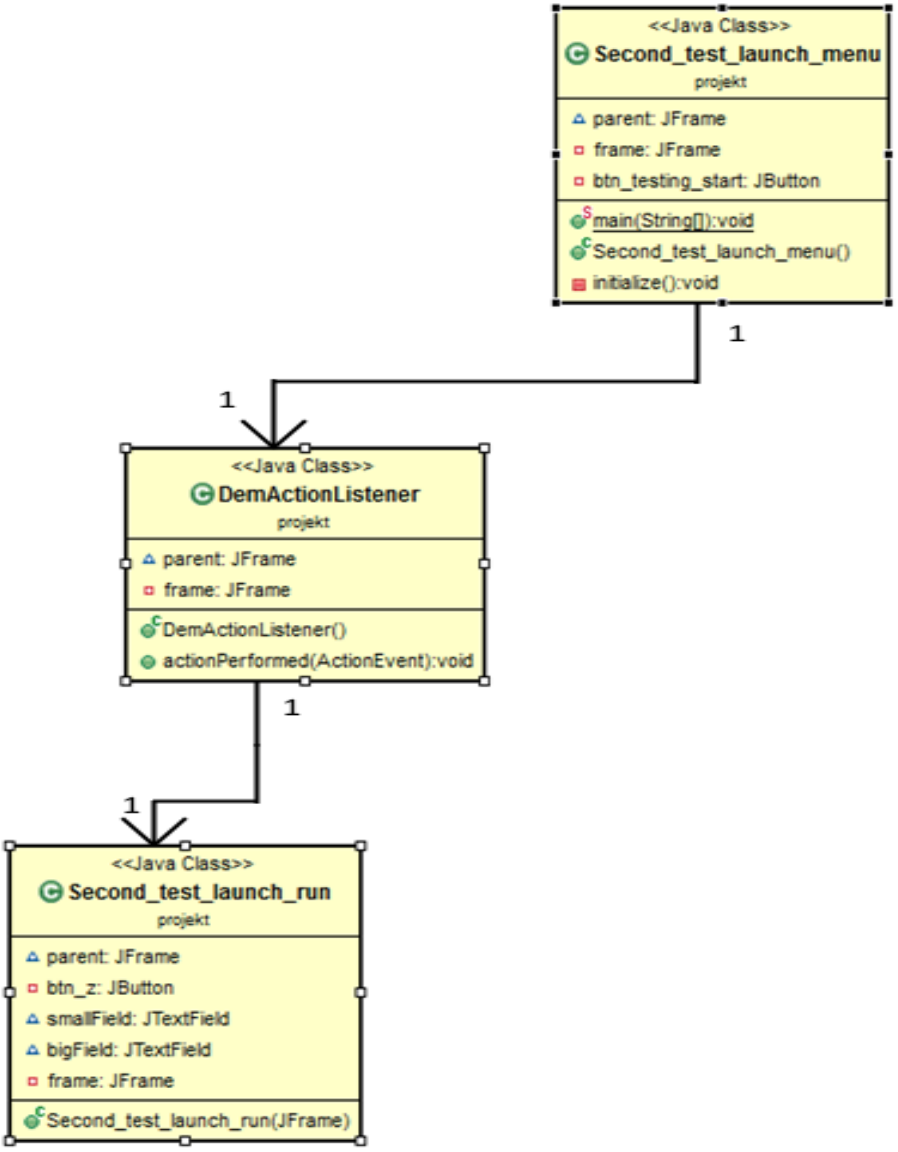

Рисунок 2.3 – Диаграмма классов программного обеспечения

Новое программное обеспечение состоит из трёх независимых друг от друга классов. В каждом из них есть ссылки на дочерний класс, а именно, при открытии программного обеспечения открывается главное меню с кнопкой «Начать», нажатие данной кнопки реализует дочерний класс, который, в свою очередь, сразу открывает свой дочерний класс, а последний уже реализует все выше рассмотренные алгоритмы.

В данное приложение можно будет добавить, в перспективе, соединение с базой данных и показ предыдущих расчётов из неё, а также, расчёт годовой нормы отходов, чтобы её не приходилось вводить вручную.

2.3 Реализация программных модулей

Для разработки приложения был выбран язык программирования Java. Java является языком программирования общего назначения, который является параллельным, на основе классов, объектно-ориентированным и специально разработан, чтобы иметь как можно меньше зависимостей реализации. Он предназначен, чтобы позволить разработчикам приложений «писать один раз, работать в любом месте» (WORA), это означает, что скомпилированный код Java может работать на всех платформах, которые поддерживают Java без необходимости перекомпиляции. Данный язык был выбран из-за достаточно хорошо понятного синтаксиса.

Синтаксис Java в значительной степени похож на C++. В отличие от C++, который сочетает в себе синтаксис для структурированного, родового, и объектно-ориентированного программирования, java была построена почти исключительно как объектно-ориентированный язык. Весь код написан внутри классов, и каждый элемент данных является объектом, за исключением примитивных типов данных (т. е. целых чисел, чисел с плавающей запятой, логических значений и символов), которые не являются объектами по соображениям производительности. Java повторно использует некоторые популярные аспекты C++ (например, метод printf).

Так же, в отличие от C++, Java не поддерживает перегрузку операторов или множественное наследование для классов, хотя множественное наследование поддерживается для интерфейсов.

Стиль javadoc комментирования позволяет пользователю запускать исполняемый файл Javadoc для создания документации для программы и может

быть прочитан некоторыми интегрированными средами разработки, такими как Eclipse, чтобы позволить разработчикам получить доступ к документации в IDE.

Рассмотрим инструменты, с которые дают возможность разрабатывать приложения на данном языке. Существует множество IDE, которые позволяют разработать Java-приложение, некоторые из них приведены в таблице 2.4.

Таблица 2.2 – Сравнение IDE

IDE	Лицензия	Windows	Linux	mac OS	Другие платформы	Разработка GUI
BEA Workshop for WebLogic	Проприетарная	Да	Да	Да		Да
BlueJ	GPL	Да	Да	Да	Solaris, Windows, Linux, Mac OS X	Нет
Eclipse	EPL	Да	Да	Да	Solaris	Да
Geany	GPL	Да	Да	Да	JVM	Нет
Greenfoot	GPL	Да	Да	Да	Solaris	Нет
NetBeans	CDDL, GPL, LGPL	Да	Да	Да	Solaris	Да
Rational Application Developer	Проприетарная	Да	Да	Да	Solaris, AIX	Да

Для разработки было решено выбрать Eclipse из-за простой установки, удобного пользовательского интерфейса и бесплатной лицензии. Рассмотрим поподробнее данную среду разработки.

Eclipse Software development kit (SDK) – это бесплатное программное обеспечение с открытым исходным кодом, выпущенное на условиях общественной лицензии Eclipse, хотя оно несовместимо с стандартной общественной лицензией GNU. Это была одна из первых IDE, запущенных под GNU Classpath, и она работает без проблем под IcedTea.

Eclipse является интегрированной средой разработки (IDE), используемой в компьютерном программировании, и является наиболее широко используемой java IDE. Она содержит базовое рабочее пространство и расширяемую подключаемую систему для настройки среды. Eclipse написана в основном на Java и его основное использование для разработки приложений Java, но он также может быть использован для разработки приложений на других языках программирования с помощью плагинов. Она также может быть использована для разработки документов с LaTeX (через плагин TeXlipse) и пакетов для программного обеспечения Mathematica.

Исходная кодовая база возникла из IBM VisualAge. Пакет средств разработки программного обеспечения Eclipse (SDK), включающий средства разработки Java, предназначен для разработчиков Java. Пользователи могут расширить свои возможности, установив плагины, написанные для платформы Eclipse, такие как наборы инструментов разработки для других языков программирования, и могут писать и вносить свои собственные модули плагинов. С момента внедрения реализации OSGi (Equinox) в версии 3 Eclipse, плагины могут быть подключены-остановлены динамически и называются (OSGI) пакетами [27].

Eclipse Public License (EPL) является основной лицензией, под которой выпускаются проекты Eclipse. Некоторые проекты требуют двойного лицензирования, для которого доступна Лицензия Eclipse Distribution License (EDL), хотя использование этой лицензии должно применяться и рассматривается в каждом конкретном случае.

Eclipse первоначально была выпущена под бесплатной общедоступной лицензией, но позже была повторно лицензирована под Eclipse Public License. Фонд свободного программного обеспечения заявил, что обе лицензии являются лицензиями свободного программного обеспечения, но несовместимы с GNU General Public License (GPL).

Также, для того, чтобы приложением было более комфортно пользоваться, был создан пользовательский интерфейс с использованием фреймвёрка Swing. Приведём краткую информацию о данном фреймвёрке. Swing — библиотека для создания графического интерфейса для программ на языке Java, который был разработан компанией Sun Microsystems.

Swing был разработан, чтобы обеспечить более сложный набор компонентов GUI, чем более ранний Abstract Window Toolkit (AWT). Swing обеспечивает внешний вид, который эмулирует внешний вид нескольких платформ, а также поддерживает подключаемый внешний вид, который позволяет приложениям выглядеть и чувствовать себя не связанными с базовой платформой. Он имеет более мощные и гибкие компоненты, чем AWT. В дополнение к знакомым компонентам, таким как кнопки, флажки и метки, Swing предоставляет несколько дополнительных компонентов, таких как панель с вкладками, области прокрутки, деревья, таблицы и списки.

В отличие от компонентов AWT, компоненты Swing не реализуются специфичным для платформы кодом. Вместо этого они полностью написаны на Java и поэтому не зависят от платформы. Термин "легкий" используется для описания такого элемента.

Swing предоставляет более гибкие интерфейсные компоненты, чем более ранняя библиотека AWT, но в отличие от последней, компоненты Swing разработаны для одинаковой кросс-платформенной работы. Компоненты же AWT повторяют интерфейс исполняемой платформы без изменений, AWT также использует только стандартные элементы ОС для отображения, то есть для каждого элемента создается отдельный объект ОС (окно). Поэтому библиотека AWT не позволяет создавать элементы произвольной формы. В ней возможно использовать только прямоугольные компоненты. Элементы управления на основе AWT всегда отображаются поверх Swing-элементов. Это происходит потому, что все Swing компоненты отображаются на поверхности контейнера.

Начиная с ранних версий Java, часть Abstract Window Toolkit (AWT) предоставляет независимые от платформы API для компонентов пользовательского интерфейса. В AWT каждый компонент визуализируется и управляется собственным одноранговым компонентом, специфичным для базовой оконной системы.

В отличие AWT, компоненты Swing часто описываются как легковесные, поскольку они не требуют выделения собственных ресурсов в windowing toolkit операционной системы. Компоненты AWT называются тяжеловесными компонентами.

Большая часть Swing API, как правило, является дополнительным расширением AWT, а не прямой заменой. На самом деле, каждый легкий интерфейс Swing в конечном счете существует в тяжеловесном компоненте AWT, потому что все компоненты верхнего уровня в Swing (JApplet, JDialog, JFrame и JWindow) расширяют контейнер верхнего уровня AWT. До обновления Java 6.10, использовать как легкие и тяжелые компоненты в одном окне был вообще запрещен из-за Z-порядок несовместимости. Однако более поздние версии Java исправили эти проблемы, и компоненты Swing и AWT теперь могут использоваться в одном GUI без проблем Z-порядка.

Основная функциональность рендеринга, используемая Swing для рисования его легких компонентов, обеспечивается Java 2D, другой частью JFC.

Стандартный набор виджетов (SWT) - это конкурирующий набор инструментов, первоначально разработанный IBM и поддерживаемый сообществом Eclipse. Реализация SWT имеет больше общего с тяжеловесными компонентами AWT. Это дает такие преимущества, как более точная точность с базовым инструментарием машинного оконного управления, за счет увеличения экспозиции к родной платформе в модели программирования.

Сильная зависимость SWT от JNI сделает его медленнее, когда компонент GUI и Java должны передавать данные, но быстрее при рендеринге, когда модель данных была загружена в GUI.

Компоненты Swing поддерживают специфические динамически подключаемые виды и поведения. Благодаря им возможна адаптация к графическому интерфейсу платформы, другими словами к компоненту можно динамически подключить другой, специфический для операционной системы. Сюда относится в том числе и созданный программистом вид и поведение. Отсюда следует, что приложения, использующие Swing, могут выглядеть как родные приложения для данной операционной системы, но основным минусом таких «легковесных» компонентов является относительно медленная работа, а положительная сторона — универсальность интерфейса созданных приложений на всех платформах.

Теперь, когда язык программирования, IDE и фреймвёрк выбраны и алгоритм расчёта составлен, приступим к реализации самого программного обеспечения.

На рисунке 2.4 показан фрагмент кода, вычисляющий χ^2.

```
        if (e.getKeyChar() == KeyEvent.VK_ENTER) {
DefaultTableModel tm = (DefaultTableModel) table2.getModel();
for (int i = 0; i < tf1; i++){
    for (int j = 0; j < tf2; j++){
obs [i][j]  = tm.getValueAt(i, j);
table2.setValueAt(obs [i][j], i, j);
System.out.print(" | "+obs [i][j]+" | ");

oc[i][j] = Double.parseDouble((String) obs [i][j]);
System.out.print(oc[i][j]);
n = oc [i][j] + n;


nx[i] = oc [i][j] + nx[j];

ny[j] = oc [i][j] + ny[j];

if (j == (tf2-1)){
    System.out.println(zz = zz + oc [i][j]);}
System.out.println("zz = "+zz);
System.out.println();
//Расчёты (χ^2)/n

        Xkwad += ((oc [i][j] * oc [i][j])/(nx[i]*ny[j]))-1;
        /*System.out.println("matrixA ["+(i+1)+"]["+(j+1)+"] = "+oc [i][j]);
        System.out.println("nx["+(i+1)+"] = "+nx[i]);
        System.out.println("ny["+(j+1)+"] = "+ny[j]);
        System.out.println("\u03C7^2/n = "+Xkwad);*/

Xkwad = n * Xkwad; //χ^2

double aXkwad = Xkwad;
/*System.out.println("---------------------------");
System.out.println("Результат:");
System.out.println("\u03C7^2 = "+Xkwad);*/

G =Math.sqrt(aXkwad/(aXkwad + 1));
}


}}
```

Рисунок 2.4 – Фрагмент кода, вычисляющий χ^2

Далее, на рисунке 2.5 показано вычисление коэффициента взаимной сопряжённости Пирсона.

```
System.out.println("\u03C7^2 = "+Xkwad);*/

G =Math.sqrt(aXkwad/(aXkwad + 1));
}


}}
```

Рисунок 2.5 – Фрагмент кода, рассчитывающий коэффициент взаимной сопряжённости Пирсона

В программном обеспечении данный коэффициент рассчитываются с помощью простых математических операций с использованием результатов предыдущих расчётов.

Приложение состоит из двух окон, первым открывается главное меню, при нажатии кнопки «Начать» открывается второе окно с пустой таблицей, в которую пользователь вводит значения, нажимает клавишу Enter и под таблицей выводится результат.

В ней есть три класса:

- «Second_test_launch_run»,
- «Second_test_launch_menu»,
- «DemActionListener».

Разберём, какие именно компоненты программного обеспечения они реализуют. Очевидно, что «Second_test_launch_menu» является реализацией главного меню, «DemActionListener» реализует открытие окна с таблицей при нажатии кнопки «Начать», а «Second_test_launch_run» - саму таблицу и расчёты с введёнными в неё данными.

Далее приведём фрагменты кодов каждого из классов. На рисунке 2.6 представлен фрагмент кода «Second_test_launch_menu».

```
10
11 public class Second_test_launch_menu {
12     JFrame parent;
13     private JFrame frame;
14     private JButton btn_testing_start;
15
16     /**
17      * Launch the application.
18      */
19     public static void main(String[] args) {
20         EventQueue.invokeLater(new Runnable() {
21             public void run() {
22                 try {
23                     Second_test_launch_menu window = new Second_test_launch_menu();
24                     window.frame.setVisible(true);
25                 } catch (Exception e) {
26                     e.printStackTrace();
27                 }
28             }
29         });
30     }
31
32     /**
33      * Create the application.
34      */
35     public Second_test_launch_menu() {
36         initialize();
37     }
38
39     /**
40      * Initialize the contents of the frame.
41      */
42     private void initialize() {
43         frame = new JFrame();
44         frame.setBounds(100, 100, 450, 300);
45         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
46         frame.getContentPane().setLayout(null);
47         frame.setLocationRelativeTo(null);
48
49
50         JLabel lbl_enter_device_aname = new JLabel("Расчёт оценки силы связи и");
51         lbl_enter_device_aname.setFont(new Font("Times New Roman", Font.BOLD, 22));
52         lbl_enter_device_aname.setBounds(12, 28, 700, 33);
53         JLabel lbl_enter_device_bname = new JLabel("коэффициентов взаимной сопряженности");
54         lbl_enter_device_bname.setFont(new Font("Times New Roman", Font.BOLD, 22));
55         lbl_enter_device_bname.setBounds(12, 28, 700, 65);
56         frame.getContentPane().add(lbl_enter_device_aname);
57         frame.getContentPane().add(lbl_enter_device_bname);
58         ActionListener actionListener = new DemActionListener();
```

Рисунок 2.6 – Фрагмент кода, реализующий класс «Second_test_launch_menu»

На рисунке 2.7 представлен фрагмент кода «DemActionListener».

```
8
9  public class DemActionListener implements ActionListener {
10     JFrame parent;
11     private JFrame frame;
12     public void actionPerformed(ActionEvent e) {
13         Second_test_launch_run main_window = new Second_test_launch_run(frame);
14         main_window.setLocationRelativeTo(null);
15         main_window.setVisible(true);
16
17         frame.setVisible(false);
18     }
19 }
20
```

Рисунок 2.7 – Фрагмент кода, реализующий класс «DemActionListener»

На рисунке 2.8 представлен фрагмент кода «Second_test_launch_run».

```
        this.parent=parent;
             // Размещение таблиц в панели с блочным расположением
            Box contents = new Box(BoxLayout.Y_AXIS);
            frame = new JFrame();
            frame.setBounds(100, 100, 450, 300);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setContentPane(contents);
            setSize(500, 400);
            setVisible(false);

        JPanel panel = new JPanel();
        String input1 = JOptionPane.showInputDialog("Введите число строк:");
        String input2 = JOptionPane.showInputDialog("Введите число столбцов:");
        String input3 = JOptionPane.showInputDialog("Введите годовую норму количества отходов:");
        int tf1 = Integer.parseInt(input1);
        int tf2 = Integer.parseInt(input2);
        double tf3 = Double.parseDouble(input3);



        // Таблица с настройками

// Заголовки столбцов

        setVisible(true);
        JTable table2 = new JTable(tf1, tf2);
        // Настройка таблицы
        table2.setRowHeight(30);
        table2.setIntercellSpacing(new Dimension(10, 10));
        table2.setGridColor(Color.black);
        table2.setShowVerticalLines(true);



        contents.add(new JScrollPane(table2));
        contents.add(new JScrollPane(panel));
        table2.addKeyListener(new KeyAdapter() {
            private double G=0;
            private Object [][] obs = new Object[tf1][tf2];
            private double [][] oc = new double [tf1][tf2];
            private double n = 0;
            private double [] nx = new double [(int)tf1];
            private double [] ny = new double [(int)tf2];
            public double zz = 0;
            public double Xkwad = 0; //χ^2
            @Override
            public void keyReleased(KeyEvent e) {
                if (e.getKeyChar() == KeyEvent.VK_ENTER) {
        DefaultTableModel tm = (DefaultTableModel) table2.getModel();
```

Рисунок 2.8 – Фрагмент кода «Second_test_launch_run»

На данном фрагменте кода показаны два массива nx и ny. В них будут записываться суммы строк и столбцов соответственно, для дальнейшего использования в расчётах. Так же переменная zz является суммой отходов, производимых в год, высчитываемая для сравнения с годовой нормой. В данную переменную записывается сумма последнего (считая слева-направо) столбца.

Число строк и столбцов в программном обеспечении вводятся пользователем. Затем пользователю следует ввести годовую норму отходов, а потом и саму матрицу, то есть, наблюдения или частоты.

Теперь рассмотрим пользовательский интерфейс программы. Окно главного меню, с диалоговым окном для ввода количества строк представлено на рисунке 2.9.

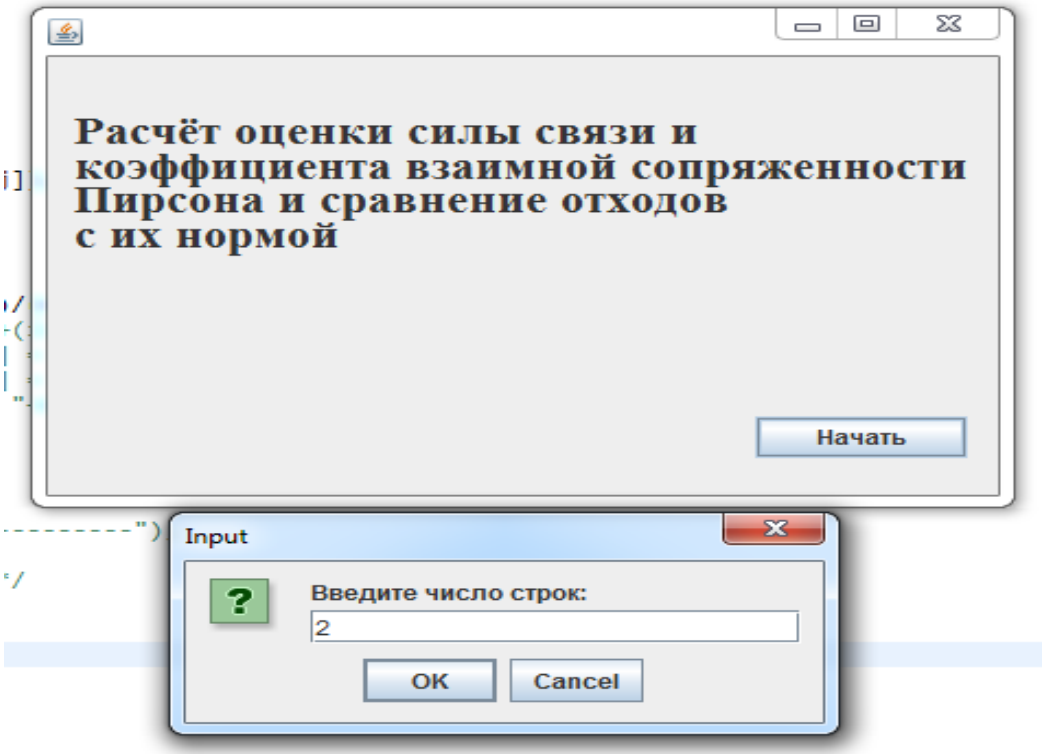

Рисунок 2.9 – Главное меню, с диалоговым окном для ввода количества строк

Диалоговое окно для ввода количества столбцов появляется следом за вводом числа строк и нажатия кнопки «OK» и является аналогичным диалоговым окном.

Рисунок 2.10, представленный ниже, показывает, что программа корректно высчитывает результат (была введена годовая норма отходов, равная 0.784).

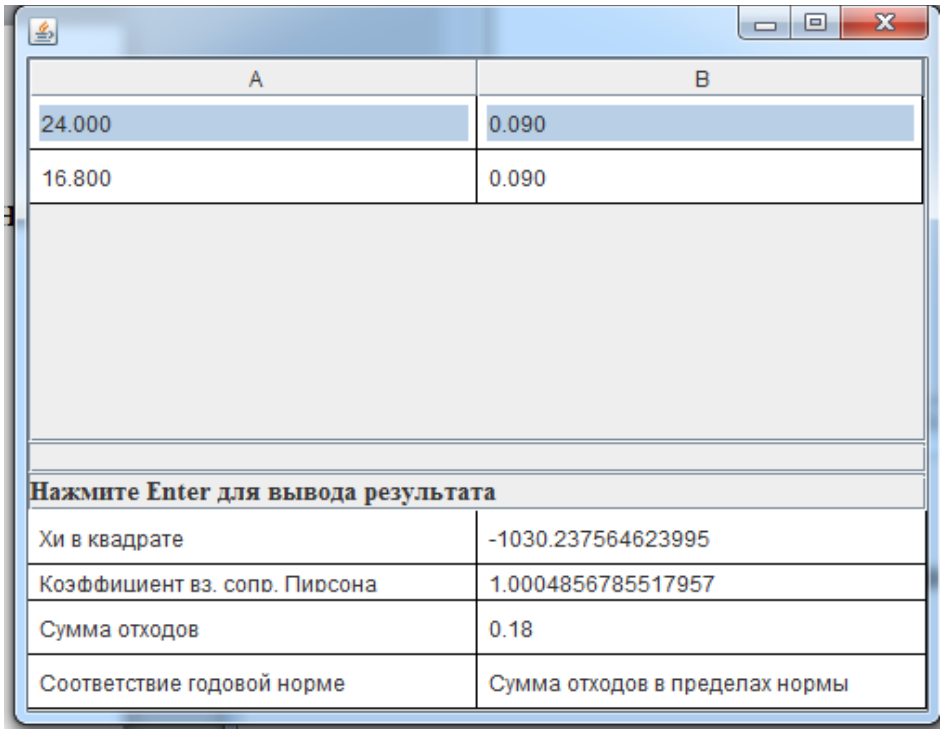

Рисунок 3.2 – Результат, посчитанный на основе данных из таблицы 2.1

В данном случае, столбец А – это количество закупаемого сырья, т/год, а В – количество образования отхода, т/год. Первая строка – это данные, показывающие количество закупленной муки и получаемых от неё отходов в год, а вторая – данные, показывающие количество сахара и производимых отходов от него в год. Как уже отмечалось, чтобы понять, каким является распределение данных о количестве отходов и сырья: зависят ли друг от друга элементы таблицы или нет, высчитывается табличное значение χ^2 и сравнивается с фактическим χ^2, значение которого находит специалист. Если $\chi^2_{факт} > \chi^2_{табл}$, то делается вывод, что элементы таблицы зависят друг от друга и можно говорить о зависимости между признаками [22]. Коэффициент Пирсона рассчитывается наряду с $\chi^2_{табл}$, чтобы более точно измерить тесноту зависимости экологических данных друг от друга (чем коэффициент Пирсона ближе к единице, тем сильнее зависимость).

3 ОТЛАДКА И ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Проведение вычислительных экспериментов

Основной целью тестирования является обнаружение сбоев программного обеспечения, чтобы дефекты могли быть обнаружены и исправлены. Тестирование не может установить, что продукт функционирует должным образом при всех условиях. Однако оно устанавливает, что программный продукт не функционирует должным образом при определенных условиях. Сфера тестирования программного обеспечения часто включает в себя изучение кода, а также выполнение этого кода в различных средах и условиях. Также она изучает аспекты кода: делает ли он то, что он должен делать, и делает то, что ему нужно делать. Сегодня в разработке программного обеспечения организация тестирования может быть отделена от команды разработчиков, и для членов группы тестирования существуют различные роли. Информация, полученная в результате тестирования программного обеспечения, может использоваться для корректировки процесса разработки программного обеспечения.

Хотя тестирование может определить правильность программного обеспечения в предположении некоторых конкретных гипотез, оно не может определить все дефекты в программном обеспечении, вместо этого оно дает критику или сравнение, которое сравнивает состояние и поведение продукта с тестовыми предсказаниями ситуаций — принципами или механизмами, с помощью которых кто-то может распознать проблему. Тестовые предсказания ситуаций могут включать (но не ограничиваться ими) спецификации, контракты, сопоставимые продукты, прошлые версии одного и того же продукта, выводы о предполагаемой или ожидаемой цели, ожидания пользователей или клиентов, соответствующие стандарты, применимое законодательство или другие критерии [28].

Каждый программный продукт создаётся для определённого заказчика, поэтому программное обеспечение для предприятий автомобилестроения полностью отличается от программного обеспечения кондитерских предприятий. Потому, когда разрабатывается программный продукт, его нужно оценить, будет ли он приемлемым для конечных пользователей, предприятия, покупателей и других заинтересованных сторон и тестирование программного обеспечения помогает сделать эту оценку.

Поскольку количество возможных тестов даже для простых программных компонентов практически бесконечно, все тестирование программного обеспечения использует некоторую стратегию для выбора тестов. Эти тесты возможны в течение доступного времени и ресурсов, и в результате тестирование программного обеспечения обычно (но не исключительно) пытается выполнить программу или приложение с целью обнаружения программных ошибок (ошибок или других дефектов). Работа по тестированию — это итерационный процесс. Потому что, когда одна ошибка исправлена, она может осветить другие, более глубокие ошибки или даже создать новые.

Тестирование программного обеспечения может предоставить объективную, независимую информацию о качестве программного обеспечения и риске его отказа пользователям или спонсорам.

Также, тестирование может проводиться, как только появится реализация программного обеспечения (даже если оно частично завершено). Общий подход к разработке программного обеспечения часто определяет, когда и как проводятся испытания, например, в поэтапном процессе большинство тестов выполняется после определения системных требований и их последующей реализации в тестируемых программах, однако, при гибком подходе требования, программирование и тестирование часто выполняются одновременно.

Одним из распространенных источников дорогостоящих дефектов являются пробелы в требованиях. Например, непризнанные требования, которые приводят к

ошибкам пропуска разработчиком программы. Пробелы требований часто могут быть нефункциональными требованиями, такими как тестируемость, масштабируемость, ремонтопригодность, удобство использования, производительность и безопасность.

Сбои программного обеспечения происходят через следующие процессы. Программист совершает ошибку (error), которая приводит к дефекту (fault, bug) в исходном коде программы, и если этот дефект выполнен, то в определенных ситуациях система даст неправильные результаты, вызывая сбой, но не все дефекты обязательно приведут к сбоям, например, дефекты в «мертвом коде» никогда не приведут к сбоям. Дефект может превратиться в сбой при изменении среды, примеры этих изменений в среде включают программное обеспечение, запускаемое на новой аппаратной платформе компьютера, изменения в исходных данных или взаимодействие с другим программным обеспечением. Одиночный дефект может привести к отказу.

Фундаментальная проблема тестирования программного обеспечения заключается в том, что тестирование при всех комбинациях входов и предварительных условий (начальное состояние) невозможно, даже с простым продуктом, и это значит, что количество дефектов в программном продукте может быть очень большим, и дефекты, которые возникают нечасто, трудно найти в тестировании. Что еще более важно, нефункциональные измерения качества (как это должно быть по сравнению с тем, что он должен делать) — удобство использования, масштабируемость, производительность, совместимость, надежность — могут быть весьма субъективными. То, что представляет достаточную ценность для одного человека, может быть невыносимо для другого.

Разработчики программного обеспечения не могут протестировать все, но они могут использовать комбинаторный дизайн теста для определения минимального количества тестов, необходимых для получения необходимого покрытия. Комбинаторный дизайн теста позволяет пользователям получить

больший охват теста с меньшим количеством тестов, ищут ли они скорость или глубину теста, они могут использовать комбинаторные методы дизайна теста для построения составленного изменения в их тестовые случаи.

Существуют различные методы тестирования программного обеспечения. Чтобы выбрать один из видов тестирования, сделаем обзор их сути.

Обзоры, пошаговые руководства или проверки являются, по сути, статическим тестированием. Фактическое же выполнение программного кода с заданным набором тестовых случаев является сутью динамического тестирования. Статическое тестирование, по сути своей, часто неявно, как корректура. Это когда инструменты программирования или текстовые редакторы проверяют структуру исходного кода или компиляторы (предкомпиляторы) проверяют синтаксис и поток данных как статический анализ программы. Динамическое тестирование происходит при запуске самой программы, и может начаться до того, как программа будет завершена на 100%, чтобы протестировать определенные разделы кода и применяются к дискретным функциям или модулям. Типичными методами для этого являются либо использование заглушек/драйверов, либо выполнение из среды отладчика.

Статическое тестирование включает проверку. Тогда как динамическое тестирование также включает проверку. Вместе они помогают улучшить качество программного обеспечения. Среди методов статического анализа можно использовать мутационное тестирование, чтобы убедиться, что тестовые случаи обнаружат ошибки, которые вводятся путем изменения исходного кода.

Тестирование может быть нескольких уровней.

Тестирование на разных уровнях производится на протяжении всего жизненного цикла разработки и сопровождения программного обеспечения, уровень тестирования определяет то, над чем производятся тесты. То есть, производятся ли над отдельным модулем, группой модулей или системой, в

целом, проведение тестирования на всех уровнях системы — это залог успешной реализации и сдачи проекта.

Рассмотрим, что представляют из себя суть каждого из уровней тестирования, имеющих названия:

- модульного тестирования;
- интеграционного тестирования;
- системного тестирования;
- приемочного тестирования.

В компьютерном программировании сутью модульного тестирования является метод тестирования программного обеспечения, с помощью которого проверяются отдельные единицы исходного кода, наборы одного или нескольких модулей компьютерной программы вместе с соответствующими контрольными данными, процедурами использования и операционными процедурами, чтобы определить, подходят ли они для использования.

Можно рассмотреть модуль как самую маленькую тестируемую часть применения, в процедурном программировании единица может быть целым модулем, но чаще это отдельная функция или процедура, в объектно-ориентированном программировании единица часто является целым интерфейсом, таким как класс, но может быть отдельным методом. Модульные тесты — это короткие фрагменты кода, созданные программистами или иногда тестировщиками белого ящика в процессе разработки. Он формирует основу для компонентного испытания [29].

В идеале каждый тестовый случай независим от других. Заменители, такие как заглушки методов, макеты объектов, подделки и тестовые жгуты, могут использоваться для помощи в изолированном тестировании модуля. Модульные тесты обычно пишутся и запускаются разработчиками программного обеспечения, чтобы убедиться, что код соответствует своей конструкции и ведет себя так, как задумано.

Поскольку некоторые классы могут иметь ссылки на другие классы, тестирование класса может часто переходить в тестирование другого класса. Распространенным примером этого являются классы, зависящие от базы данных: для тестирования класса тестировщик часто пишет код, который взаимодействует с базой данных. Это ошибка, потому что модульный тест обычно не должен выходить за пределы своей собственной границы класса, и особенно не должен пересекать такие границы процесса/сети, потому что это может ввести неприемлемые проблемы производительности в набор модульных тестов. Пересечение таких границ единиц измерения превращает модульные тесты в интеграционные тесты, и в случае сбоя таких тестов может быть неясно, какой компонент вызывает сбой. Вместо этого разработчик программного обеспечения должен создать абстрактный интерфейс вокруг запросов к базе данных, а затем реализовать этот интерфейс со своим собственным макетом объекта.

Сутью интеграционного тестирования является то, что это этап тестирования программного обеспечения, на котором отдельные программные модули объединяются и тестируются как группа. Это происходит после модульного тестирования и проверки до тестирования. Интеграционное тестирование использует в качестве входных модулей модульное тестирование, группирует их в более крупные агрегаты, применяет тесты, определенные в плане интеграционного тестирования, к этим агрегатам и предоставляет в качестве выходных данных интегрированную систему, готовую к тестированию системы.

Рассмотрим, также, суть некоторых типов интеграционного тестирования: big-bang, top-down.

В подходе big-bang («большого взрыва»), большая часть из начатых модулей соединена совместно для того чтобы сформировать полную программную систему или большую часть системы и после этого использована для интеграционного тестирования. Этот метод очень эффективен для экономии времени в процессе интеграционного тестирования. Однако, если тестовые случаи и их результаты не

будут записаны должным образом, весь процесс интеграции будет более сложным и может помешать команде тестирования достичь цели интеграционного тестирования.

Сутью тестирования top-down («снизу вверх») является подход к интегрированному тестированию, при котором сначала тестируются компоненты самого низкого уровня, а затем используются для облегчения тестирования компонентов более высокого уровня. Процесс повторяется до тех пор, пока не будет проверен компонент в верхней части иерархии.

Все нижние или низкоуровневые модули, процедуры или функции интегрированы и после этого испытаны. После интеграционного тестирования интегрированных модулей нижнего уровня будет сформирован следующий уровень модулей, который может быть использован для интеграционного тестирования. Этот подход полезен только тогда, когда все или большинство модулей одного уровня разработки готовы. Этот метод также помогает определить уровни разработанного программного обеспечения и облегчает отчет о ходе тестирования в виде процента.

Системное тестирование программного или аппаратного обеспечения, по сути своей, это тестирование, проводимое на полной интегрированной системе для оценки соответствия системы заданным требованиям. Системное тестирование входит в сферу тестирования черного ящика и как таковое не должно требовать знания внутреннего дизайна кода или логики.

Как правило, системное тестирование принимает в качестве входных данных все «интегрированные» программные компоненты, которые прошли интеграционное тестирование, а также саму программную систему, интегрированную с любой применимой аппаратной системой (системами). Цель интеграционного тестирования состоит в том, чтобы обнаружить любые несоответствия между программными блоками, которые интегрированы вместе (называемые сборками) или между любой сборкой и оборудованием. Системное

тестирование-это более ограниченный вид тестирования; оно направлено на выявление дефектов как в рамках «межсборки», так и в рамках системы в целом.

Приемочное тестирование используется для проведения эксплуатационной готовности (предварительного выпуска) продукта, услуги или системы в рамках системы менеджмента качества. Оно является распространённым типом нефункционального тестирования программного обеспечения, используемое в основном в проектах разработки и сопровождения программного обеспечения. Этот тип тестирования фокусируется на эксплуатационной готовности системы, которая будет поддерживаться, и/или стать частью производственной среды. Следовательно, это также известно как тестирование эксплуатационной готовности или эксплуатационная готовность и тестирование гарантии. Функциональное тестирование в рамках приемочного тестирования ограничено теми тестами, которые необходимы для проверки нефункциональных аспектов системы.

Теперь разберём сути методов чёрного и белого ящика. Тестирование белого ящика (также известное как тестирование прозрачного ящика, тестирование стеклянного ящика, тестирование прозрачного ящика и структурное тестирование), по сути, является методом тестирования программного обеспечения, который проверяет внутренние структуры или работу приложения, в отличие от его функциональности (т. е. тестирование черного ящика). В тестировании белого ящика внутренняя работа системы, а также навыки программирования используются для разработки тестовых случаев. Тестировщик выбирает входные сигналы для того чтобы работать пути через код и определять предполагаемые выходы. Это аналогично тестированию узлов в цепи, например, тестирование в цепи (ICT). Тестирование белого ящика можно приложить на уровнях блока, внедрения и системы процесса испытания программного обеспечения. Хотя традиционно принято говорить о тестировании белого ящика как о выполняемом на уровне модулей, сегодня он чаще используется для

интеграции и тестирования системы. Он может тестировать пути внутри блока, пути между блоками во время интеграции и между подсистемами во время теста системного уровня. Хотя этот метод дизайна теста может расчехлить много ошибки или проблем, он имеет потенциал пропустить нереализованные части спецификации или отсутствующих требований.

Тестирование чёрного ящика рассматривает функциональность без каких-либо знаний о внутренней реализации, не видя исходный код. Тестировщики знают только о том, что программное обеспечение должно делать, а не о том, как оно это делает. Черного ящика методы тестирования включают: разделение эквивалентности, граничные анализ стоимости, все – тестирование пар, изменения состояния таблиц, тестирования таблицы решений, пуха тестирования, модель тестирования, использовать тестирование, исследовательское тестирование, и Спецификация на основе тестирования.

Тестирование на основе спецификаций направлено на тестирование функциональности программного обеспечения в соответствии с применимыми требованиями. Этот уровень тестирования обычно требует предоставления тестировщику подробных тестовых случаев, который затем может просто проверить, что для данного входа выходное значение (или поведение) либо «is», либо «is» не совпадает с ожидаемым значением, указанным в тестовом случае. Тестовые случаи строятся вокруг спецификаций и требований, то есть, что приложение должно делать. Он использует внешние описания программного обеспечения, включая спецификации, требования и проекты для получения тестовых случаев. Эти тесты могут быть функциональными или нефункциональными, хотя обычно функциональными.

Тестирование на основе спецификаций может быть необходимо для обеспечения правильной функциональности, но этого недостаточно для защиты от сложных или рискованных ситуаций.

Одним из преимуществ метода черного ящика является то, что не требуется никаких знаний программирования. Какие бы предубеждения ни были у программистов, тестер, скорее всего, имеет другой набор и может подчеркнуть различные области функциональности. Поскольку они не изучают исходный код, бывают ситуации, когда тестировщик пишет много тестовых случаев, чтобы проверить то, что могло быть протестировано только одним тестовым случаем, или оставляет некоторые части программы непроверенными.

Этот метод тестирования может применяться на всех уровнях тестирования: модульное, интеграционное, системное и приемочное. Он обычно включает в себя большинство, если не все тестирование на более высоких уровнях, но также может доминировать модульное тестирование.

В тестировании программного обеспечения сутью тестирования на отказ и восстановление является проверка того, насколько хорошо приложение может восстанавливаться после сбоев, сбоев оборудования и других подобных проблем. Другими словами это – процесс тестирования, в котором приложение или система подвергается воздействию экстремальных условий или моделируемых условий, чтобы вызвать сбой, например, сбои ввода-вывода устройства или недопустимые указатели и ключи базы данных. Далее, при отладке программы, процессы восстановления и приложение или система контролируется и проверяется для проверки надлежащего применения или системы, и восстановление данных было достигнуто [30].

В данном тестировании создаётся принудительный сбой программного обеспечения различными способами для проверки правильности выполнения восстановления. Тестирование на отказ и восстановление не следует путать с тестированием надежности, которое пытается обнаружить конкретную точку, в которой происходит сбой. Тестирование на отказ и восстановление делается для того, чтобы проверить, как быстро и лучше, приложение может восстановить

против любого вида аварии или сбоя оборудования и так далее. Тип или степень восстановления указывается в спецификациях требований.

Приведём примеры тестирования на отказ и восстановление:

- в то время как приложение работает, вдруг перезагрузить компьютер, а затем проверьте валидность целостности данных приложения;
- ввести некорректные данные (например, ввести буквы туда, куда допускается вводить только числа).

Таблицы 3.1-3.2 наглядно показывают суть рассмотренных выше уровней и видов тестирования.

Таблица 3.1 – Уровни тестирования программного обеспечения

Признаки	Уровни тестирования программного обеспечения				
	Статическое тестирование	Динамическое тестирование	Модульное тестирование	Интеграционное тестирование	Системное тестирование
Тестируется	Функционал программы	Функционал программы	Модули или компоненты	Группа модулей	Все интегрированные компоненты
Знание исходного кода	Не требуется	Не требуется	Требуется	Требуется	Не требуется
Суть	Обзоры, пошаговые руководства или проверки	Фактическое выполнение программного кода с заданным набором тестовых случаев	Тестирование отдельных единиц исходного кода	Модули собираются в группу, которая тестируется. big-bang – все модули сразу, top-down – сначала модули низкого уровня, затем – высокого	Тестирование, проводимое на полной интегрированной системе для оценки соответствия системы заданным требованиям

Таблица 3.2 – Виды тестирования программного обеспечения

Признаки	Виды тестирования программного обеспечения		
	Метод чёрного ящика	Метод белого ящика	Тестирование на отказ и восстановление
Тестируется	Вся программа	Вся программа	Вся программа
Знание исходного кода	Не требуется	Требуется	Не требуется

Суть	Тестирование по спецификациям	Проверяет внутренние структуры или работу приложения	Проверка отказоустойчивости программы

Рассмотрев сути многих методов тестирования на некоторые из видов тестирования, был сделан вывод, что для разрабатываемого программного обеспечения наиболее подходящим является системный уровень, а видом – тестирование на отказ и восстановление. Итак, разобравшись, какой вид и уровень тестирования подходит для разрабатываемого программного обеспечения, приступим непосредственно к тестированию.

Проведём вычислительные эксперименты с разработанным программным обеспечением. Для этого, попробуем ввести 2 строки и 2 столбца. Далее заполняем таблицу, введя в одну из ячеек отрицательное число. Рисунок 3.1 показывает реакцию программного обеспечения. В качестве годовой нормы количества отходов введено случайное число, в экспериментальных целях и равно 5.

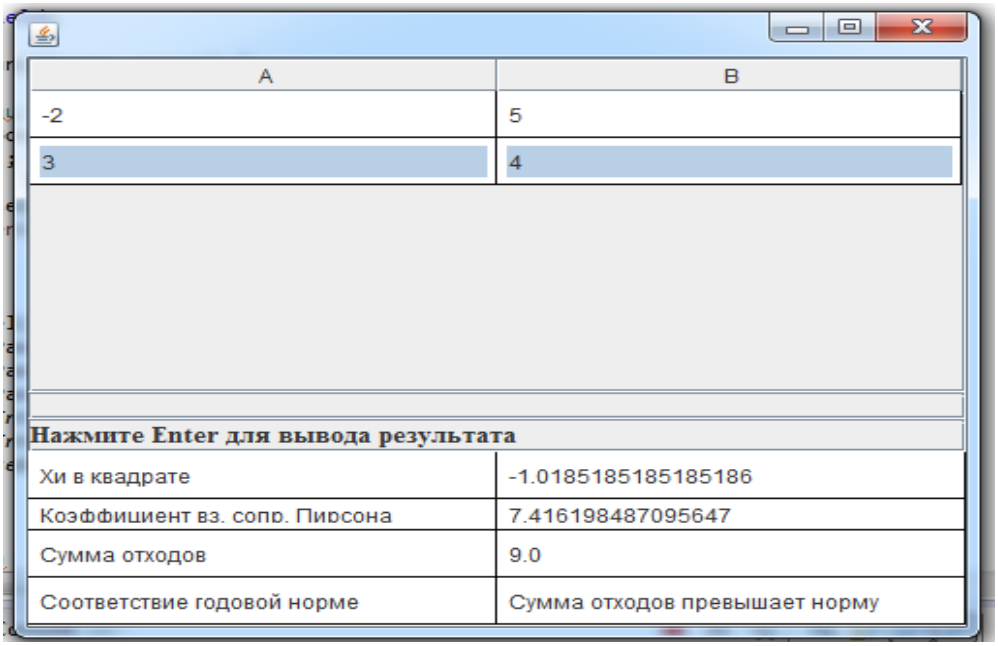

Рисунок 3.1 – Реакция программы на ввод отрицательного числа в таблицу

Из данного рисунка видно, что программа корректно посчитала и вывела результат.

Далее был проведён тест, выявляющий, как функционирует главное меню и кнопка «Начать». В данном компоненте приложения всё стабильно функционирует.

Если же мы введём в количество строк и столбцов буквы, то программа не вылетает. Это показано на рисунке 3.2.

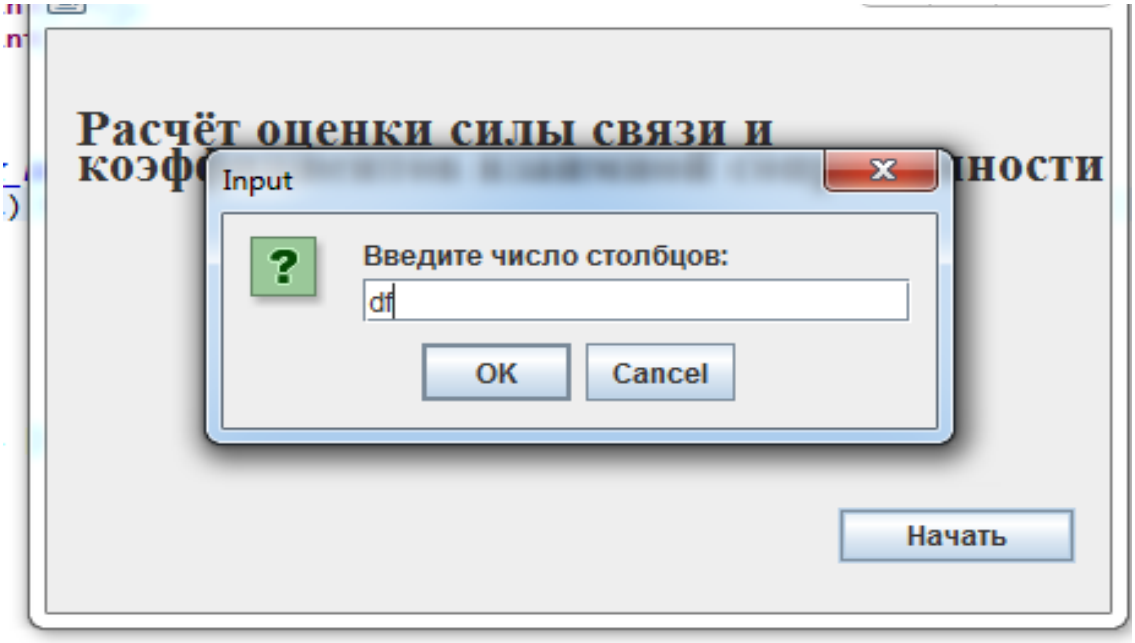

Рисунок 3.2 – Ввод букв в поле ввода количества столбцов

Программа не зависла и не закрылась. При нажатии на кнопку «Начать», она предлагает заново ввести количество строк и столбцов.

3.2 Корректировка разработанного программного обеспечения для анализа экологических данных

Но если ввести в таблицу не цифры а буквы, то программа не будет реагировать. Это исправляется оператором if, который будет проверять введённые данные. Представлено на рисунке 3.3.

```
69          String character = (String) obs [i][j];
70
71          char c = character.charAt(0);
72
73          if (Character.isDigit(c)) {
74
75          } else if (Character.isLetter(c)) {
76              JOptionPane.showMessageDialog(null, "Вы ввели букву в таблицу!");
77
78          }
```

Рисунок 3.3 – Фрагмент кода, реализующий вывод сообщение о введённой букве

Реализация вывода показана на рисунке 3.4.

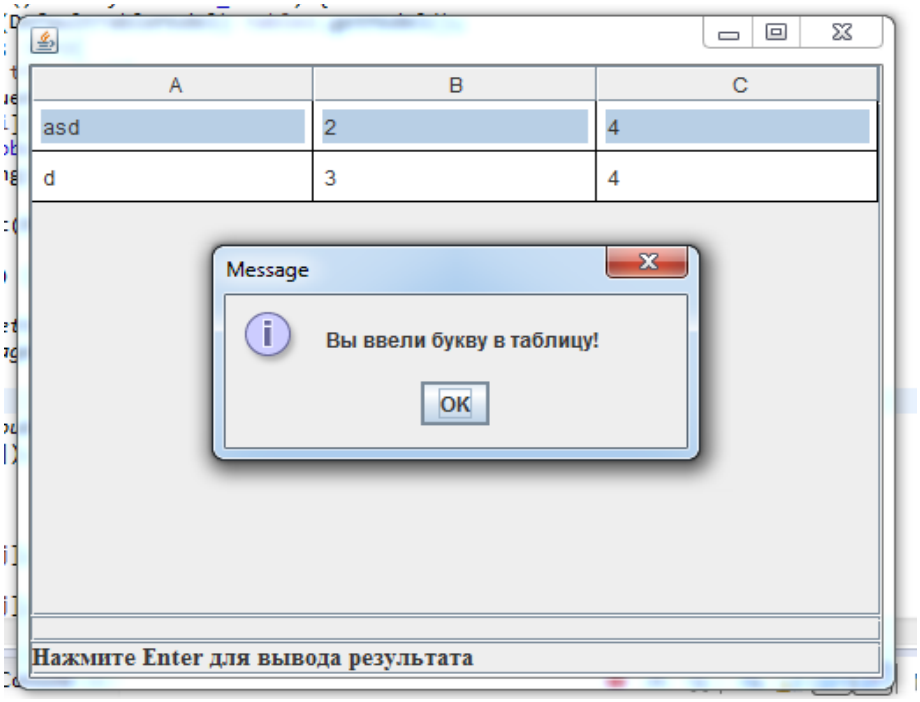

Рисунок 3.4 – Вывод сообщения о введённой в таблицу букве

В ходе выполнения программы получены результаты расчёта критерия $\chi^2_{табл}$, сравнивая который с $\chi^2_{факт}$ определяется случайность или неслучайность распределения экологических данных, а также, рассчитываются коэффициент взаимной сопряженности Пирсона, чтобы точнее измерить тесноту зависимости.

ЗАКЛЮЧЕНИЕ

В результате сравнительного анализа различных IDE, фреймвёрков и видов тестирования, для выявления их преимуществ и недостатков, для разработки и тестирования программного обеспечения были выбраны IDE Eclipse, фреймвёрк Swing и тестирование на отказ и восстановление.

Разработанное программное обеспечение получены результаты расчёта критерия $\chi^2_{табл}$, сравнивая который с $\chi^2_{факт}$ определяется случайность или неслучайность распределения экологических данных, а также, рассчитываются коэффициенты взаимной сопряженности, чтобы измерить тесноту зависимости, то есть, чтобы понять, каким является распределение данных о количестве отходов и сырья: зависят ли друг от друга элементы таблицы или нет, высчитывается табличное значение χ^2. Если $\chi^2_{факт} > \chi^2_{табл}$, то делается вывод, что элементы таблицы зависят друг от друга и можно говорить о зависимости между признаками. Для измерения тесноты данной зависимости используют коэффициенты взаимной сопряженности, которые рассчитываются, используя результаты расчёта оценки, при помощи несложных математических операций.

Разработанное приложение состоит из двух окон. Первым открывается главное меню, при нажатии кнопки «Начать» программное обеспечение предлагает пользователю ввести количество строк и столбцов в диалоговых окнах. Далее открывается второе окно с пустой таблицей, в которую пользователь вводит значения, нажимает клавишу Enter и под таблицей выводится результат.

Также было проведено тестирование разработанного программного обеспечения. В результате тестирования в программу был добавлен вывод сообщения при вводе в таблицу сопряжённости букв.

В данное приложение можно будет добавить соединение с базой данных и показ предыдущих расчётов из неё, а также, расчёт годовой нормы отходов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. — СПб.: Питер, 2014. — 320 с. — ISBN 5-94723-698-2.

2. Булгаков Н.Г., Дубинина В.Г., Левич А.П., А.Т.Терехин. // Известия РАН. Сер. Биол. 1995. №2. 218 с.

3. Герберт Шилдт. Java 8. Полное руководство, 9-е издание Java 8. The Complete Reference, 9th Edition. — М.: «Вильямс», 2015. — 1376 с.

4. Герберт Шилдт. SWING: руководство для начинающих = SWING: A BEGINNER'S GUIDE. — М.: «Вильямс», 2015. — С. 704. — ISBN 0-07-226314-8.

5. Гленфорд Майерс, Том Баджетт, Кори Сандлер. Искусство тестирования программ, 3-е издание = The Art of Software Testing, 3rd Edition. — М.: «Диалектика», 2015. — 272 с. — ISBN 978-5-8459-1796-6.

6. Замолодчиков Д.Г., Булгаков Н.Г., Гурский А.Г., Левич А.П., Чесноков С.В. К методике применения детерминационного анализа для обработки экологических данных // Биол. науки. 1992. №7. 116-133 с.

7. Иван Портянкин. Swing: Эффектные пользовательские интерфейсы, 2-е издание. — 2-е. — Санкт-Петербург: «Лори», 2014. — С. 600. — ISBN 978-5-85582-305-9.

8. К изучению опасности загрязнения биосферы: воздействие додецилсульфата натрия на планктонных фильтраторов // ДАН. 2017, Т. 425, No. 2, 271—272 с.

9. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование. — М.: «Вильямс», 2016. — 374 с. — ISBN 5-8459-0336-X.

10. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. — Киев: ДиаСофт, 2016. — 544 с. — ISBN 9667393879.

11. Кей С. Хорстманн. Java SE 8. Вводный курс = Java SE 8 for the Really Impatient. — М.: «Вильямс», 2017. — 208 с.

12. Кузенкова Г. В. Введение в экологический мониторинг: учебное пособие. — Н. Новгород: НФ УРАО, 2014. — 72 с.

13. Кучай Л.А., Соколова Е. Применение метода детерминационного анализа и экологического нормирования для оценки состояния водной экосистемы // Вода: химия и экология. 2013. - № 4 (58). С. 13-18.

14. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2015. — 464 с. — (Addison-Wesley Signature Series). — 1000 экз. — ISBN 978-5-8459-1625-9.

15. Левич А.П., Терехин А.Т // Водные ресурсы. 1997. №3. 328 с.

16. Максимов В.Н., Булгаков Н.Г., Милованова Г.Ф., Левич А.П. Детерминационный анализ в экосистемах: сопряженности для биотических и абиотических компонентов // Известия РАН. Сер. Биол. 2014. № 4. 482 с.

17. Максимов В.Н., Левич А.П., Булгаков Н.Г., Милованова Г.Ф. Детерминационный анализ связей в биотическом и абиотическом компонентах экосистемы и между этими компонентами // Известия РАН. Сер. биол. 1999.

18. Муртазов А. К. Экологический мониторинг. Методы и средства: Учебное пособие. Часть 1 / А. К. Муртазов; Рязанский государственный университет им. С. А. Есенина. — Рязань, 2017. — 146 с.

19. Реагирование тест-организмов на загрязнение водной среды четвертичным аммониевым соединением // Водные ресурсы. 1991. № 2. 112—116 с.

20. Руководство по гидробиологическому мониторингу пресноводных экосистем. Под. ред. Абакумова В.А. СПб: Гидрометеоиздат, 1992.318 с.

21. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. — М.: БИНОМ, 2015. — 368 с. — ISBN 978-5-94774-825-3.

22. Снытко В. А., Собисевич А. В. Концепция геоэкологического мониторинга в трудах академика И. П. Герасимова // География: развитие науки и образования. — Т. 1. — Изд-во РПГУ имени Герцена Санкт-Петербург, 2017. — 88-91 с.

23. Фрэд Лонг, Дхрув Мохиндра, Роберт С. Сикорд, Дин Ф. Сазерленд, Дэвид Свобода. Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищённых программ = Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs. — М.: «Вильямс», 2014. — 256 с.

Литература на иностранном языке

24. David M. Geary: Graphic Java 2, Volume 2: Swing, Prentice Hall, ISBN 0-13-079667-0

25. Matthew Robinson, Pavel Vorobiev: Swing, Second Edition, Manning, ISBN 1-930110-88-X

26. Meyer, Bertrand (August 2008). "Seven Principles of Software Testing" (pdf). Computer. Vol. 41 no. 8. pp. 99–101. doi:10.1109/MC.2008.306. Retrieved November 21, 2017.

27. John Zukowski: The Definitive Guide to Java Swing, Third Edition, Apress, ISBN 1-59059-447-9

28. James Elliott, Robert Eckstein, Marc Loy, David Wood, Brian Cole: Java Swing, O'Reilly, ISBN 0-596-00408-7

29. Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour: The JFC Swing Tutorial: A Guide to Constructing GUIs, Addison-Wesley Professional, ISBN 0-201-91467-0

30. Ostroumov S.A. Biological Effects of Surfactants. CRC Press. Taylor & Francis. Boca Raton, London, New York. 2015. 279 p.

ПРИЛОЖЕНИЕ А. КЛАСС «SECOND_TEST_LAUNCH_MENU»

```
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class Second_test_launch_menu {
JFrame parent;
private JFrame frame;
private JButton btn_testing_start;
/**
* Launch the application.
*/
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
public void run() {
try {
Second_test_launch_menu window = new Second_test_launch_menu();
window.frame.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
} }
});
}
/**
* Create the application.
*/
```

```
public Second_test_launch_menu() {
initialize();   }
/**
* Initialize  the contents of the frame.
*/
private void initialize()  {
frame = new JFrame();
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().setLayout(null);
frame.setLocationRelativeTo(null);
JLabel lbl_enter_device_aname  = new JLabel("Расчёт оценки силы связи и");
lbl_enter_device_aname.setFont(new  Font("Times  New  Roman",  Font.BOLD,
22));
lbl_enter_device_aname.setBounds(12, 28, 700, 33);
JLabel  lbl_enter_device_bname  =  new  JLabel("коэффициентов  взаимной
сопряженности");
bl_enter_device_bname.setFont(new  Font("Times  New  Roman",  Font.BOLD,
22));
lbl_enter_device_bname.setBounds(12, 28, 700, 65);
frame.getContentPane().add(lbl_enter_device_aname);
frame.getContentPane().add(lbl_enter_device_bname);
ActionListener actionListener = new DemActionListener();
btn_testing_start = new JButton("Начать");
btn_testing_start.addActionListener(actionListener);
btn_testing_start.setBounds(323, 215, 97, 25);
frame.getContentPane().add(btn_testing_start);
}}
```

ПРИЛОЖЕНИЕ Б. КЛАСС «DEMACTIONLISTENER»

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
public class DemActionListener implements ActionListener {
JFrame parent;
private JFrame frame;
public void actionPerformed(ActionEvent e) {
Second_test_launch_run main_window = new Second_test_launch_run(frame);
main_window.setLocationRelativeTo(null);
main_window.setVisible(true);
frame.setVisible(false);
}}
```

ПРИЛОЖЕНИЕ В. КЛАСС «SECOND_TEST_LAUNCH_RUN»

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author Student
 */
public class Second_test_launch_run extends JFrame
{
JFrame parent;
private JButton btn_z;
    JTextField smallField,  bigField;
    private JFrame frame;
    public Second_test_launch_run(JFrame parent) {
        this.parent=parent;
          // Размещение таблиц в панели с блочным расположением
           Box contents = new Box(BoxLayout.Y_AXIS);
         frame = new JFrame();
           frame.setBounds(100, 100, 450, 300);
           frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           setContentPane(contents);
           setSize(500, 400);
```

```
            setVisible(false);

        JPanel panel = new JPanel();
        String input1 = JOptionPane.showInputDialog("Введите число строк:");
        String input2 = JOptionPane.showInputDialog("Введите число столбцов:");
        String input3 = JOptionPane.showInputDialog("Введите годовую норму количества отходов:");
        int tf1 = Integer.parseInt(input1);
        int tf2 = Integer.parseInt(input2);
        double tf3 = Double.parseDouble(input3);
        // Таблица с настройками
        setVisible(true);
        JTable table2 = new JTable(tf1, tf2);
        // Настройка таблицы
        table2.setRowHeight(30);
        table2.setIntercellSpacing(new Dimension(10, 10));
        table2.setGridColor(Color.black);
        table2.setShowVerticalLines(true);
        contents.add(new JScrollPane(table2));
        contents.add(new JScrollPane(panel));
        table2.addKeyListener(new KeyAdapter() {
          private double G=0;
            private Object [][] obs = new Object[tf1][tf2];
            private double [][] oc = new double [tf1][tf2];
            private double n = 0;
            private double [] nx = new double [(int)tf1];
            private double [] ny = new double [(int)tf2];
            public double zz = 0;
```

```
    public double Xkwad = 0; //χ^2
        @Override
    public void keyReleased(KeyEvent e) {
        if (e.getKeyChar() == KeyEvent.VK_ENTER) {
 DefaultTableModel tm = (DefaultTableModel) table2.getModel();
 for (int i = 0; i < tf1; i++){
  for (int j = 0; j < tf2; j++){
 obs [i][j]  = tm.getValueAt(i, j);
 table2.setValueAt(obs [i][j], i, j);
 System.out.print(" | "+obs [i][j]+" | ");
String character = (String) obs [i][j];

char c = character.charAt(0);

if (Character.isDigit(c)) {

} else if (Character.isLetter(c)) {
    JOptionPane.showMessageDialog(null, "Вы ввели букву в таблицу!");

}
oc[i][j] = Double.parseDouble((String) obs [i][j]);
System.out.print(oc[i][j]);
 n = oc [i][j] + n;
        nx[i] = oc [i][j] + nx[j];
        ny[j] = oc [i][j] + ny[j];
        if (j == (tf2-1)){
            System.out.println(zz = zz + oc [i][j]);}
        System.out.println("zz = "+zz);
```

```
            System.out.println();
            //Расчёты (χ^2)/n

            Xkwad += ((oc [i][j] * oc [i][j])/(nx[i]*ny[j]))-1;
            /*System.out.println("matrixA  ["+(i+1)+"]["+(j+1)+"] = "+oc [i][j]);
            System.out.println("nx["+(i+1)+"] = "+nx[i]);
            System.out.println("ny["+(j+1)+"] = "+ny[j]);
            System.out.println("\u03C7^2/n  = "+Xkwad);*/
            Xkwad = n * Xkwad; //χ^2

            double aXkwad = Xkwad;
            /*System.out.println("-------------------------");
            System.out.println("Результат:");
            System.out.println("\u03C7^2  = "+Xkwad);*/

  G =Math.sqrt(aXkwad/(aXkwad + 1));
            }}
String str, astr, bstr, cstr;
cstr = "";
str = Double.toString(Xkwad);
astr = Double.toString(G);
bstr = Double.toString(zz);
if (zz <= tf3){
  cstr = "Сумма отходов в пределах нормы";
} else if (zz > tf3){
            cstr = "Сумма отходов превышает норму";
  }
```

```
Object[][] array = new String[][] {{ "Хи в квадрате" , str}, { "Коэффициент вз. сопр. Пирсона"  , astr}, { "Сумма отходов "  , bstr}, { "Соответствие годовой норме"  , cstr},};
    // Заголовки столбцов
        Object[] columnsHeader = new String[] {"_", "_"};
        JTable table3 = new JTable(array, columnsHeader);
        // Настройка таблицы
        table3.setRowHeight(30);
        table3.setRowHeight(1, 20);
        table3.setIntercellSpacing(new Dimension(10, 10));
        table3.setGridColor(Color.black);
        table3.setShowVerticalLines(true);
        contents.add(table3);
        setContentPane(contents);
        setSize(500, 400);
        setVisible(true);
                }}
        });
        JLabel lbl_enter_device_aname = new JLabel("Нажмите Enter для вывода результата");
        lbl_enter_device_aname.setFont(new Font("Times New Roman", Font.BOLD, 14));

        lbl_enter_device_aname.setBounds(323, 300, 97, 25);
        contents.add(new JScrollPane(lbl_enter_device_aname));}}
```