

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 ОБЗОР ЛИТЕРАТУРЫ	8
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ	16
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	21
3.1 Модуль интерфейса пользователя	21
3.2 Модуль описания команд	22
3.3 Модуль журналирования	25
3.4 Модуль настройки приложения	27
3.5 Модуль хранения данных	28
3.6 Модуль анализа файлов	31
3.7 Модуль пересылки сообщений	34
3.8 Модуль управления правилами.	36
3.9 Модуль взаимодействия с монитором	40
3.10 Модуль мониторинга файловой системы	42
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ	45
4.1 Алгоритм работы функции process_received_events	45
4.2 Алгоритм работы функции process_possible_pair_event	47
4.3 Алгоритм работы метода run потока приёма и передачи сообщений ...	49
4.4 Алгоритм работы метода get_rules	52
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ	55
5.1 Модульное тестирование	56
5.2 Интеграционное тестирование	57
5.3 Полное функциональное тестирование программы	59
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	62
6.1 Требования к аппаратному и программному обеспечению	62
6.2 Руководство по установке системы	62
6.3 Руководство по использованию программного средства	63
7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ФАЙЛОВОГО МЕНЕДЖЕРА С ФУНКЦИЕЙ АВТОМАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ФАЙЛОВ	66
7.1 Характеристика программного продукта	66
7.2 Расчёт сметы затрат и отпускной цены программного продукта	67
7.3 Расчёт экономического эффекта ПО от свободной реализации на рынке	71
7.4 Расчёт показателей экономической эффективности разработки и реализации программного продукта	72
ЗАКЛЮЧЕНИЕ	75
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	76
ПРИЛОЖЕНИЕ А	77
ПРИЛОЖЕНИЕ Б	79

ВВЕДЕНИЕ

Задачей данного дипломного проекта является создание программного приложения, которое позволит решить задачу автоматического поддержания структуры каталогов на жёстком диске пользователя в необходимом для него порядке.

На данный момент существует множество программ так или иначе упрощающих задачу поддержания структуры каталогов, содержащейся на компьютере пользователя. К этим программам относятся торрент-клиенты и менеджеры загрузок, позволяющие выбрать куда сохранять файлы, сортировщики, предоставляющие такие возможности как быстрое перемещение файлов определённых типов в определённую директорию и групповое переименование или удаление файлов, подпадающих под определённый шаблон, каталогизаторы, позволяющие создавать коллекции из файлов и предоставляющие быстрый поиск по коллекциям, программы позволяющие найти и удалить временные файлы, созданные другими программами и т.д.

В рамках решения задачи по поддержанию структуры каталогов на жёстком диске пользователя существующие программы имеют некоторые недостатки: для одних программ такая функциональность является побочной, другие требуют периодического вовлечения пользователя, третьи имеют очень узкую направленность. Ни одна из них не работает по принципу межсетевого экрана, когда пользователь настраивает необходимые правила в самом начале, и программа работает по этим правилам до тех пор, пока пользователю не понадобится их изменить, решая возникающие задачи в автоматическом режиме. Наличие приложения, работающего подобным образом, могло бы снять с пользователя необходимость время от времени наводить порядок в своих файлах и каталогах.

Таким образом, целью данного дипломного проекта является создание программного приложения, способного автоматически определять попадающие на жёсткий диск файлы, определять под какие правила, заданные пользователем, этот файл подпадает и в зависимости от этого правила произвести над этим файлом определённое действие. Поскольку настройка данной системы, используя непосредственно термины файловой системы, может оказаться для пользователя сложной – необходимо создание дополнительного уровня абстракции над файловой системой, на котором пользователю будет предоставлен набор команд, скрывающих для него эту сложность.

1 ОБЗОР ЛИТЕРАТУРЫ

В настоящее время проблема превращения жёсткого диска компьютера в файловую свалку появляется практически у каждого пользователя. Минутное желание скачать файл, создание одноразовых каталогов без последующего удаления, копирование файлов с накопителей, нежелание создавать структуру директорий для своего жёсткого диска или нежелание этой структуры придерживаться – всё это приводит к тому, что жёсткий диск пользователя превращается в беспорядочное нагромождение файлов и папок, и нахождение нужной информации на диске превращается в нетривиальную задачу. В настоящем дипломном проекте рассматривается программное приложение, способное поддерживать первоначально созданную пользователем структуру каталогов без его участия, в автоматическом режиме.

Файлы могут попадать на жёсткий диск пользователя разными путями: скачивание файлов из интернета, обмен документами с помощью USB-накопителей, файлы, создаваемые приложениями во время работы, файлы, которые создаёт сам пользователь. Все эти файлы потенциально вызывают нарушение первоначальной структуры каталогов пользователя, особенно в течении большого количества времени. Что бы показать масштаб проблемы нужна статистика файлового обмена. Статистику по файловому обмену на USB носителях и созданию файлов пользователей найти невозможно, поскольку эта информация никем не собирается. Но статистика, позволяющая сказать насколько часто файлы качают из интернета - находится в открытом доступе.

Наибольшее количество файлов попадает на жёсткий диск пользователя из сети Internet. По статистике Google Trends запросы со словом «скачать» имеют стабильную популярность. Она колеблется в районе 75/100 и достигает пика во время новогодних праздников 90-100/100 (см. рисунок 1.1). Больше всего таких запросов приходится на страны СНГ. Наибольшая популярность – в Казахстане 89/100 (см. рисунок 1.2) [1]. По статистике Яндекс, поисковик ежемесячно выдаёт 210 628 770 ответов на запросы со словом «скачать» [2]. Анализ аналогичных запросов в англоязычных странах показывает, что пользователи из этих стран качают файлы чаще чем аналогичные пользователи из стран СНГ. По статистике того же Google Trends популярность запроса со словом «download» колеблется в районе 75-100/100 (см. рисунок 1.3). Среди англоязычных стран распределение частоты запросов на скачивание по регионам также не равномерно. Чаще всего запросы приходят из стран с большим количеством населения, на которое приходится большое количество доступной компьютерной техники. Также на частоту скачиваний влияет правовая культура населения страны и эффективность методов борьбы с пиратством. Этим обуславливается тот факт, что несмотря на широкую доступность компьютерной техники и высокую компьютерную грамотность, население стран Европы и Северной Америки качает файлы гораздо реже чем население стран Ближнего Востока и Южной Азии.

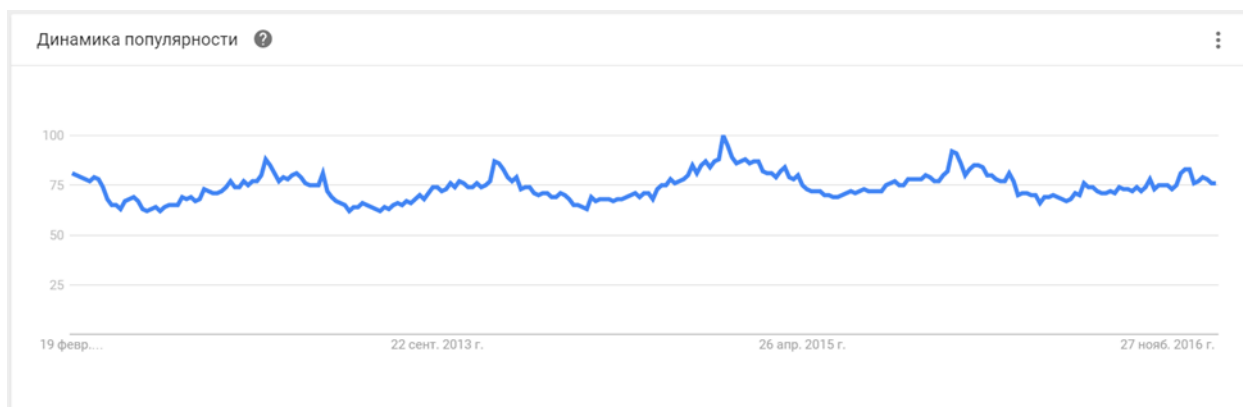


Рисунок 1.1 - Статистика популярности запросов со словом «скачать»

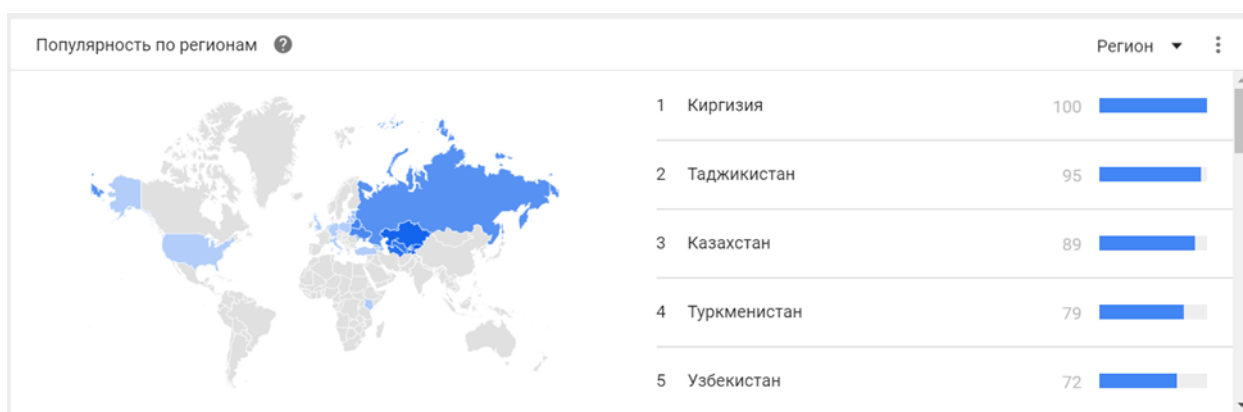


Рисунок 1.2 - Популярность запросов со словом «скачать» по регионам

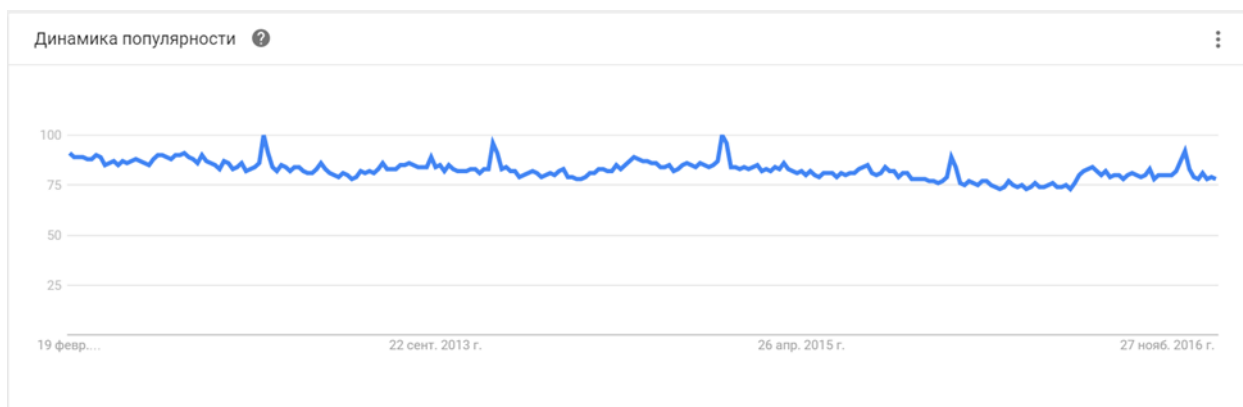


Рисунок 1.3 - Статистика популярности запросов со словом «download»

Наиболее популярны такие запросы в Индии, Пакистане и Индонезии. Статистику частоты запросов со словом скачать в англоговорящих странах можно увидеть на рисунке 1.4. [3].

Также об активности обмена файлами можно судить по активности использования торрент-клиентов по миру. Об этом можно судить по статистике, собранной компанией EZTV, специализирующейся на TV-торрентах. Согласно этой статистике, самым популярным клиентом является китайский «Xunlei» а самыми активными пользователями торрентов оказались китайцы - каждый третий пользователь торрент-клиента.

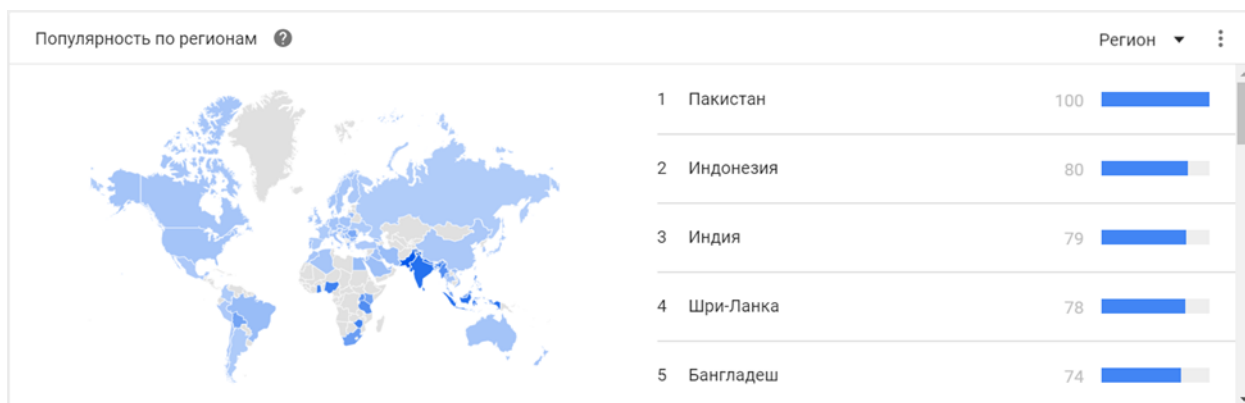


Рисунок 1.4 - Популярность запросов со словом «download» по регионам

В итоговой таблице собраны 357 миллионов уникальных пиров (равноправный участник одноранговой сети) по разным торрент-клиентам (см. таблицу 1.1) [4].

Таблица 1.1 – Статистика использования торрент-клиентов по всему миру

Место	Название клиента	Уникальных пиров
1	Xunlei	104,717,836
2	uTorrent	92,086,035
3	Azureus	86,055,354
4	Mainline	17,207,544
5	BitComet	14,341,918
6	Transmission	11,637,110
7	BitSpirit	7,132,119
8	FlashGet	3,882,628
9	TuoTu	2,531,472
10	Libtorrent	1,822,833
11	Lphant	1,437,921
12	BitTyrant	794,218
13	Xtorrent	791,925
14	rTorrent	702,498
15	QQDownload	580,103
16	Opera	556,902

Как видно из статистики, обмен файлами сегодня проходит в огромных масштабах. Это значит, что типичному пользователю компьютера довольно часто приходится сталкиваться с проблемой поиска необходимых файлов на своём компьютере. Файлы, которые необходимо найти, потенциально попали на диск от дня до месяца назад, и были при этом сохранены в разные директории или сохранены в директорию, в которую скачиваются все без исключения файлы. Однократный поиск файла в такой ситуации может занять немного времени. При многократном поиске в большом скоплении файлов и папок, которое когда-то было структурой каталогов пользователя – будет

затрачиваться значительное количество времени и сил.

Эта проблема существует уже давно. Время от времени появляются программы, позволяющие привести структуру каталогов пользователя в необходимый для него порядок. В программы, предназначенные для скачивания файлов, часто добавляются возможности, позволяющие этот порядок поддерживать.

По тому, насколько эффективно программа решает задачу поддержания в порядке структуры каталогов пользователя, программы, которые имеют такую функциональность, можно условно разделить на три категории:

- 1) Программы, позволяющие управлять сохранением файлов.
- 2) Программы, позволяющие разово упорядочить файлы в отдельно взятом каталоге пользователя.
- 3) Программы позволяющие долговременно сохранять структуру каталогов пользователя.

К программам первой категории можно отнести многочисленные менеджеры загрузки, торрент-клиенты, установщики программ, редакторы текста, музыки, видео и т.д. Эти программы, кроме своей основной функциональности, дают пользователю возможность управлять сохранением файлов, с которыми работает программа, тем самым позволяет пользователю сохранить его структуру каталогов. Сам механизм управления сохранением в таких программах может быть реализован по-разному: в одних программах файл можно сохранять разово, и с каждым новым файлом приходится заново выбирать путь, по которому он будет сохранён, в других программах путь можно указать в настройках. С другой стороны, эта же функциональность позволяет пользователю сохранять свои файлы куда попало, что производит обратный эффект – постепенное захламление жёсткого диска компьютера. В качестве примера таких программ можно привести всем известный менеджер загрузок µTorrent и текстовый редактор Notepad++ [5, 6].

К программам второй категории относятся многочисленные сортировщики файлов, программы, позволяющие удалить неиспользуемые временные файлы и программы, удаляющие файлы имеющие копии в разных каталогах.

Принцип работы с программами сортировщиками сводится к следующему – пользователь выбирает каталог, в котором он хочет отсортировать файлы, задаёт правила сортировки, которые отличаются от программы к программе, но в основном сводятся к указаниям куда переместить файлы определённого типа. Типичным примером программы-сортировщика файлов можно назвать DropIt [7]. Пример интерфейса программы DropIt, в котором настраиваются правила сортировки можно увидеть на рисунке 1.6.

В качестве примера программы, позволяющей найти и удалить неиспользуемые временные файлы, файлы, хранящиеся в кэше браузеров, файлы, имеющие копии в других каталогах и так далее можно привести программу CCleaner [8].

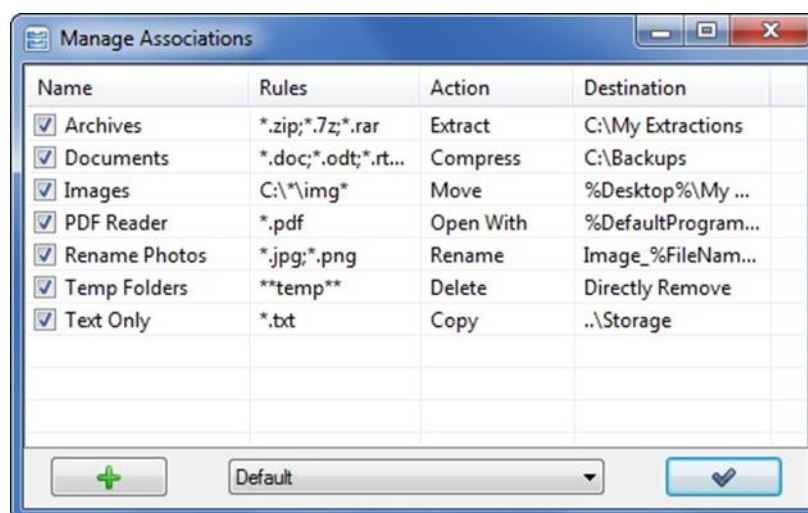


Рисунок 1.6 – Настройка правил сортировки в программе DropIt

Принцип работы с такими программами как CCleaner прост – пользователь отмечает что ему нужно, нажимает на кнопку, а программа дальше всё делает автоматически. Пример интерфейса окна поиска и удаления ненужных файлов на рисунке 1.7. Список программ, позволяющих искать и удалять файлы у которых есть копии в других директориях: DupKiller, DuplicateCleaner, CloneSpy, DuplicateFinder и т.д [9]. Пользователь задаёт какой-то критерий поиска (см. рисунок 1.8), программа ищет дубликаты, предлагает какой из N дубликатов удалить, и пользователь удаляет только то, что действительно должно быть удалено.

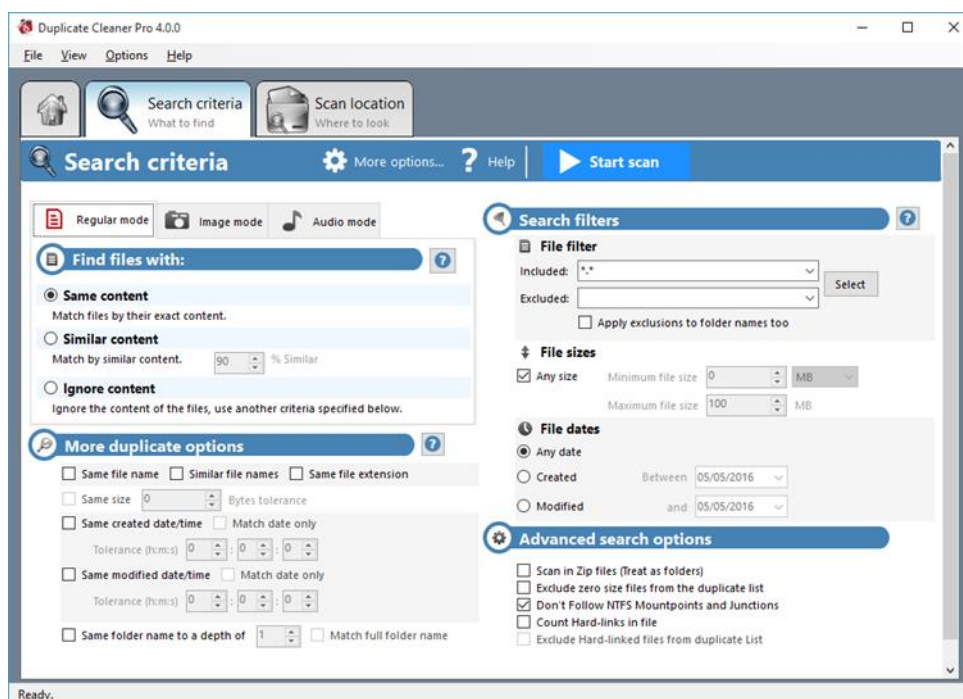


Рисунок 1.7 – Пример работы программы CCleaner

Программы, относящиеся ко второй категории могут помочь пользователю довольно быстро восстановить первоначальную структуру его

каталогов. Минус таких программ в том, что они решают проблему воссоздания исходной структуры каталогов пользователя, а не её сохранения. Используют эти программы обычно, когда структура уже нарушена и не понятна пользователю, и нахождение нужных файлов начинает занимать много времени. Плюс же в том, что такие программы решают обычно какую-то одну задачу, а не пытаются делать всё сразу, поэтому, когда у пользователя появляется конкретная проблема, ему не нужно тратить часы что бы научиться пользоваться этой программой – пользователь её запускает, программа обрабатывает, и он может спокойно работать дальше.

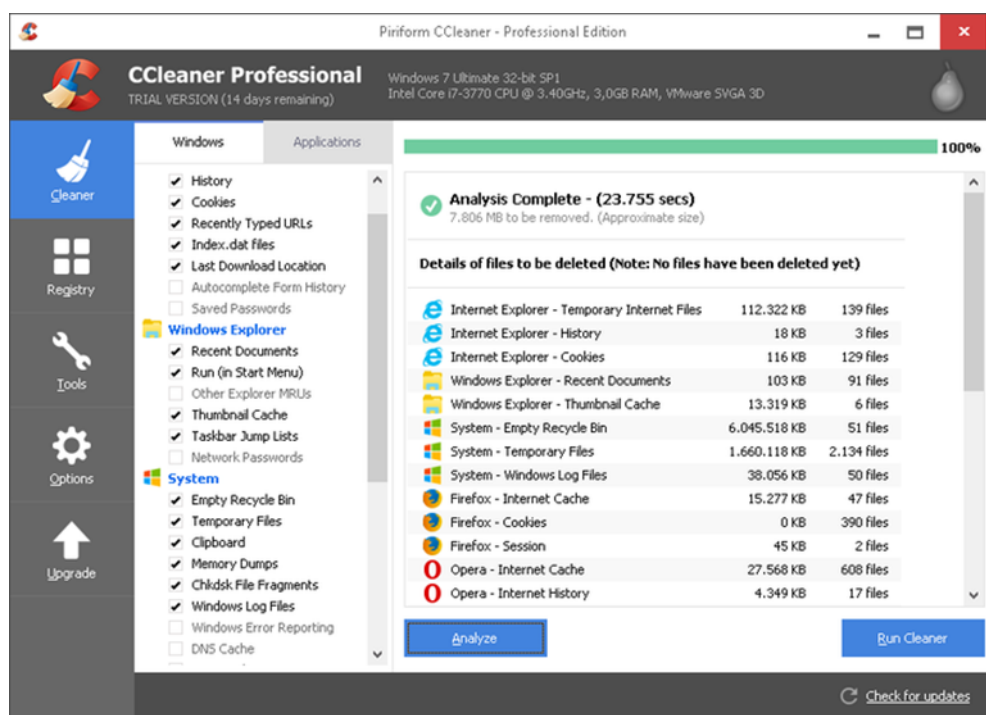


Рисунок 1.8 – Пример настройки критериев поиска в программе Duplicate Cleaner

Если программы, подпадающие под первую категорию, имеют функциональность, позволяющую пользователю поддерживать порядок на рабочем диске, программы второй категории помогают этот порядок восстанавливать, то программы третьей категории умеют этот порядок поддерживать автоматически. На данный момент среди таких программ чаще всего встречаются решения, созданные программистами для своих собственных нужд и не получившие широкое распространение. Исключением является только программа File2Folder, которая работает только на OS X, и доступна для скачивания в Mac App Store [10], далее она будет рассмотрена в качестве аналога данного дипломного проекта. Ещё одним примером может служить программа Automator, которая создана как попытка автоматизировать сортировку папки загрузок, и работает только для OS X [11].

Рассмотрим подробнее программу File2Folder. Согласно описанию, размещённому на Mac App Store, программа умеет перемещать, копировать и

удалять файлы по расширениям и части имени и переименовывать файлы согласно правилам, но в отличие от программ-сортировщиков есть возможность настроить таймер, который через заданный промежуток времени будет проверять заданные пользователем директории и распределять новые файлы в соответствии с заданными правилами.

Автоматическое выполнение каких-то действий через определённые промежутки времени – идея не новая, таким образом управляются многие задачи, которые требуют много ресурсов и времени, например, выполнение резервного копирования данных, сборка мусора в приложениях с автоматическим управлением памятью, обработка данных поступивших в базу данных за день. Но в задаче автоматического распределения файлов у этого подхода есть свои плюсы и минусы.

К плюсам подхода можно отнести простоту реализации, гибкость в управлении таймерами, которая позволяет настроить короткие промежутки, для часто используемых директорий и длинные, для директорий, которые просто нужно изредка чистить. Так же к плюсам можно отнести интуитивную понятность подхода для пользователя.

К минусам же можно отнести то, что пользователю нужно дожидаться срабатывания таймера, чтобы увидеть файл в нужной директории, что не всегда удобно. К примеру, в ситуации, когда пользователь качает музыку, для того что бы скинуть её на плеер, по окончании загрузки файлов пользователь будет ожидать что вся музыка уже распределена по папкам, но в случае, когда таймер срабатывает раз в 5 минут – это будет не так, так как половина музыки может быть распределена, а другая половина дожидаться следующего таймера. Второй минус в том, что происходит большое количество холостых срабатываний программы если в директорию не добавляется никаких файлов, что не позволяет выставить очень короткий промежуток срабатывания таймера даже для часто используемых директорий. Так же это повышает требования к экономии программой ресурсов компьютера, так как частое выполнение проверок программой, без оптимизации программы, может привести к «зависанию» компьютера пользователя, что само-собой не приемлемо.

Альтернативой подходу с таймерами может стать подход с использованием механизмов файловой системы позволяющих отслеживать события, которые в ней происходят. К инструментам, позволяющим отслеживать такие события можно отнести библиотеку inotify в ОС Linux и OS X [12] и часть библиотеки WIN64 API в ОС Windows [13].

Плюсы подхода с отслеживанием событий в том, что нет лишних срабатываний и приложение будет обрабатывать сразу после того как файлы попали на диск. То есть такой подход компенсирует все минусы подхода с таймерами.

Что касается минусов – появляется проблема, связанная со срабатываниями событий. Дело в том, что то, что файл был создан в файловой системе ещё не значит, что все данные для этого файла полностью

скопированы. В таком случае начало сортировки до полного копирования данных может привести к непредвиденным ошибкам и повреждённым данным, что недопустимо. Поэтому в рамках дипломного проекта необходимо создать механизм, способный эту ситуацию обрабатывать. Вторым минусом заключается в том, что автоматическая сортировка файлов сразу после попадания их на диск, увеличивает количество операций с диском, которых будет и так много, если файлы копируются, например, с внешнего накопителя. Если процесс копирования и процесс сортировки будут пытаться оперировать информацией большей чем пропускная способность диска – это приведёт к существенному замедлению обоих процессов. Поэтому для того, чтобы эффективно использовать диск, лучше использовать комбинированный подход, и использовать события и таймеры в зависимости от целевого предназначения директории.

Кроме задачи автоматического отслеживания файлов, и задачи абстрагирования этого отслеживания от операционной системы, в рамках данного дипломного проекта необходимо решить задачу создания абстракции над файловой системой, для предоставления пользователю списка команд, предназначенных для управления отслеживанием и распределением файлов. Наличие таких команд позволит создать единообразный интерфейс (API) для разных операционных систем, который будет использоваться на самом высоком уровне данного дипломного проекта, и который позволит пользователям создавать в будущем свои программы на его основе. Примером такой абстракции с набором команд к ней, может служить абстракция репозитория, созданная Линусом Торвальдсом в программе git [14].

Для уровня отслеживания и распределения файлов в данном дипломном проекте используется язык Python для Linux и язык C для Windows, которые хорошо подходят для обеспечения необходимого быстродействия и обеспечивают доступ к низкоуровневым функциям операционных систем [15].

На уровне API используется язык Python, для обеспечения переносимости между разными операционными системами и средами. Для обеспечения работы этого уровня в определённом окружении нужно будет только установить интерпретатор и нужные библиотеки [16].

Для уровня пользовательского интерфейса могут быть использованы разные технологии, которые будут зависеть от операционной системы, это может быть, как консоль в ОС Linux, так и WPF приложение в ОС Windows. При наличии полного API выбор технологии будет вестись по таким параметрам как простота создания интерфейса для программиста и его удобство для конечного пользователя.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

В структуре разрабатываемого дипломного проекта можно выделить следующие блоки (см. схему ГУИР.400201.170 С1):

- блок интерфейса пользователя;
- блок описания команд;
- блок настройки приложения;
- блок журнала событий;
- локальная база данных;
- блок управления правилами;
- блок мониторинга файловой системы;
- блок пересылки сообщений;
- блок анализа файлов;
- блок взаимодействия с монитором;

Блок интерфейса пользователя предназначен для предоставления пользователю возможности конфигурировать работу приложения, настраивать правила сортировки и перемещения для пользовательских директорий и просматривать отчёты о результатах работы приложения. Так же блок интерфейса пользователя позволяет своевременно выводить сообщения об ошибках, произошедших во время работы приложения. Результатом работы пользователя с данным блоком является команда в определённом формате, которая передаётся блоку описания команд, для анализа и выполнения.

Блок описания команд предназначен для приёма команд от интерфейса пользователя и их интерпретации. Он необходим для того что бы работа приложения не зависела от вида интерфейса пользователя. Блок описания команд предоставляет следующие команды:

- создание, чтение, обновление и удаление настроек приложения;
- создание, чтение, обновление и удаление правил управления файлами в пользовательских директориях;
- просмотр записей в журнале событий;
- немедленное выполнение заданного действия для заданной директории;
- откат состояния отслеживаемых директорий до заданной записи в журнале событий;
- импорт и экспорт настроек приложения;
- импорт и экспорт правил по управлению файлами;
- импорт и экспорт настроек и правил вместе;
- запуск и отключение мониторинга;

Блок описания команд взаимодействует с блоком управления правилами, блоком настройки приложения и блоком журнала событий. К блоку журнала событий имеется только доступ на чтение.

Блок настройки приложения управляет действиями, которые не

относятся к основной функциональности приложения, а влияют на его работу в конкретной операционной системе. К таким действиям относится настройка запуска отдельных блоков приложения как демонов или сервисов в операционной системе, проверка наличия всех необходимых библиотек, установка и удаление необходимых переменных окружения, проверка обновлений и т.д. Так же с помощью этого блока пользователь имеет возможность настроить специальное поведение приложения, которое будет учитывать низкую скорость жёсткого диска компьютера или высокую частоту файловых операций. Блок настройки приложения сохраняет все параметры, необходимые для работы, в файле, в специальном формате, и имеет возможность экспортировать или импортировать этот файл. Данный блок принимает изменения со стороны блока описания команд и используется блоком взаимодействия с монитором.

Локальная база данных представляет собой реляционную базу данных с высокими требованиями к скорости записи. Скорость чтения значительной роли не играет, так как в блоках, которые используют базу данных и для которых скорость чтения является критичным параметром реализован кэш.

Блок журнала событий предназначен для конвертации событий, созданных приложением, в записи журнала и сохранения этих записей в локальную базу данных. Данный блок позволяет пользователю получить информацию о том, что приложение сделало с файлами, которые подпали под правила, заданные пользователем. Так же, поскольку записи событий сохраняются последовательно, есть возможность отменить некоторые (не все) действия, сделанные приложением после определённой записи. Это даёт приложению дополнительную надёжность, так как в случае сбоя или непредвиденного поведения пользователь получает возможность вмешаться в то, что сделало приложение и исправить ошибку. События, которые служат основой для записей журнала генерируются блоком взаимодействия с монитором файловой системы.

Блок управления правилами предоставляет глобальный интерфейс по работе с правилами обработки файлов. Основная его функциональность - вычисление правила или группы правил, которые применимы к файлу, который сгенерировал событие в файловой системе. Основные ситуации, которые должны обрабатываться данным блоком:

- 1) Корректная обработка приоритетов правил. Правила с большей специфичностью должны применяться в первую очередь, а в некоторых ситуациях полностью перекрывать правила, для которых свойства подпадающих под них файлов описаны в более общем виде.

- 2) Корректная обработка пересечений правил. Если к файлу применимы два правила, которые предусматривают одинаковое действие – это действие должно примениться только один раз.

- 3) Оптимизация правил. Если за правилами, которые предусматривают какие-либо действия над файлом, идёт правило, которое должно этот файл безвозвратно удалить – правила перед удалением удаляются.

4) При удалении отслеживаемой директории, удаляются все правила, связанные с этой директорией.

5) При изменении имени отслеживаемой директории изменяются все правила, связанные с этой директорией.

6) При появлении правил, которые добавляют новую директорию для отслеживания или при удалении всех правил, связанных с одной из существующих директорий, данный блок должен оповестить монитор файловой системы, используя для этого интерфейс блока взаимодействия с монитором.

Также этот блок полностью ответственен за хранение правил и знает, как и когда их нужно добавлять, обновлять или удалять и содержит в себе уровень кэша над локальной базой данных, в который правила считываются на старте приложения. Уровень кэша необходим для того что бы обеспечить необходимую скорость обработки событий файловой системы. Потеря данных о заданных правилах не допускается, поэтому кэш сквозной - запись правила в кэш приводит к немедленной записи в базу данных.

Блок управления правилами используется блоком взаимодействия с монитором файловой системы и блоком описания команд. В качестве входной информации, по которой нужно определить правила, применимые к файлу, используются данные о событии в файловой системе и данные от блока анализа файлов. Новые правила так же добавляются в специальном формате.

Блок мониторинга файловой системы представляет собой программу в двух версиях – в виде демона Linux или сервиса Windows. Эта программа производит мониторинг файловой системы для отслеживаемых директорий, и когда в файловой системе происходит какое-то событие она конвертирует его в сообщение текстового формата и отправляет его через блок пересылки сообщений блоку взаимодействия с монитором. Данным блоком отслеживаются следующие события:

- появление нового файла в отслеживаемой директории;
- изменение имени файла в отслеживаемой директории;
- изменение имени отслеживаемой директории (приводит к обновлению правил для этой директории);
- удаление файла из отслеживаемой директории;
- изменение файла в отслеживаемой директории;
- файл был перемещён из отслеживаемой директории;
- файл был помещён в отслеживаемую директорию;
- отслеживаемая директория была удалена (приведёт к отмене всех правил, связанных с данной директорией);
- отслеживаемая директория была перемещена (приведёт к изменению всех правил, связанных с данной директорией);

Так же данный блок способен принимать сообщения о появлении новых директорий для отслеживания от основного приложения (конкретно – блок взаимодействия с монитором). Для мониторинга событий используется

библиотека inotify для ОС Linux, и функции WIN64 API для ОС Windows.

Так как блок мониторинга файловой системы работает как отдельная программа, ей нужно средство связи с основным приложением. В качестве этого средства связи выступает блок пересылки сообщений. Блок пересылки сообщений представляет собой библиотеку, предоставляющую возможность обмена сообщениями в рамках распределённого приложения. В данном дипломном проекте для этой цели будет использована библиотека ZeroMQ. Для обмена сообщениями эта библиотека использует сокет без гарантированной доставки сообщений.

Блок анализа файлов предназначен для получения всех метаданных файла, сгенерировавшего событие в отслеживаемой директории. Правила, задаваемые пользователем, могут быть привязаны к свойствам этих метаданных. К интересующим приложению свойствам файлов относятся:

- 1) Имя, расширение, дата создания (базовый анализ).
- 2) Исполнитель, альбом, жанр (анализ музыкальных файлов).
- 3) Тематика, язык, жанр (анализ электронных книг).

Результат анализа файла вместе с данными о событии служат основной информацией для определения правил, которые в свою очередь определяют набор необходимых действий над данным файлом. Блок анализа файлов используется только блоком взаимодействия с монитором файловой системы.

Блок взаимодействия с монитором файловой системы является центральным блоком приложения. Он предназначен для передачи команд блоку мониторинга файловой системы и обработки событий в отслеживаемых директориях, пришедших от этого блока.

Команда конвертируется в текстовый формат и передаётся через блок передачи сообщений. Передаются следующие команды:

- включение монитора;
- выключение монитора;
- добавление новой директории для отслеживания с настройками отслеживания;
- удаление директории из списка отслеживаемых;

События приходят от блока мониторинга также в текстовом формате. Порядок обработки события следующий:

- 1) Блок получает сообщение от блока мониторинга файловой системы через блок пересылки сообщений.
- 2) Блок создаёт отдельный поток обработки сообщения, и передаёт ему сообщение на обработку.
- 3) Сообщение переводится из текстового формата в объектный формат приложения.
- 4) По типу события определяется может ли приложение обработать его прямо сейчас и нужен ли анализ файла, сгенерировавшего событие.
- 5) Если анализ файла нужен он проводится с помощью блока анализа файлов.

6) Далее данные о событии и, опционально, результаты анализа файлов отправляются блоку управления правилами. В ответ приходит список правил, применимый к данному файлу.

7) Блок по порядку выполняет действия, предусмотренные правилами.

8) Данные о событии и действиях записываются в журнал событий.

9) Поток обработки сообщения освобождает ресурсы.

Также, поскольку этот блок является центральным и выполняет все действия над файлами, для него очень важна правильная обработка ошибок, поскольку всего одна необработанная ошибка может остановить работу всего приложения.

В этом разделе были подробно рассмотрены все основные логические блоки приложения и связи между ними, которые явно описывают то как пользователь будет взаимодействовать с программой и как компоненты программы будут взаимодействовать между собой. Система спроектирована так, что на данном этапе не зависит от целевой операционной системы и языка программирования, который может быть использован для реализации данного приложения.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе будут подробно рассмотрены все аспекты дальнейшего функционирования программы. Для этого будет проведён анализ основных модулей, из которых состоит программа и рассмотрим их зависимости. Будут подробно рассмотрены классы и их методы, функции, списки констант и основные механизмы взаимодействия между модулями программы.

В программе можно условно выделить десять модулей:

- 1) Модуль интерфейса пользователя.
- 2) Модуль описания команд.
- 3) Модуль журналирования.
- 4) Модуль настройки приложения.
- 5) Модуль хранения данных.
- 6) Модуль анализа файлов.
- 7) Модуль пересылки сообщений.
- 8) Модуль управления правилами.
- 9) Модуль взаимодействия с монитором.
- 10) Модуль мониторинга файловой системы.

Все модули данного приложения оформлены в виде пакетов языка Python. Классы, константы и функции внутри каждого из этих модулей можно поделить на сущности, обеспечивающие работу данного модуля и сущности, позволяющие другим модулям с ним взаимодействовать. Всё что можно делать с каждым модулем чётко определено интерфейсом этого модуля и\или протоколами взаимодействия. Все интерфейсы и протоколы будут так же описаны в данном разделе.

3.1 Модуль интерфейса пользователя

Данный модуль предназначен для обеспечения взаимодействия пользователя с основным приложением. Главной же задачей интерфейса пользователя с программной точки зрения является получение команды от пользователя, преобразование её в строку JSON и отправка её через сокет ZeroMQ модулю описания команд. Такое разделение позволяет реализовывать интерфейс пользователя на разных операционных системах с помощью наиболее подходящего для этих целей языка программирования.

В рамках данного дипломного проекта реализован консольный интерфейс пользователя. Реализован он в исполняемом модуле `terminal_client`. Функции, отвечающие за его реализацию:

1) `_parse_args` – функция, которая не принимает ничего и возвращающая словарь с полученными от пользователя аргументами и флагами. Аргументы командной строки, введённые пользователем, достаются стандартным модулем `argparse` из коллекции `sys.argv`, после чего

преобразуются в вид словаря по заранее заданным правилам.

2) `_get_command_json_string` – функция, которая принимает на вход словарь с аргументами и возвращающая команду, сериализованную в строку JSON.

3) `main` – функция, которая последовательно вызывает две предыдущие команды, и отправляет через интерфейс обмена сообщениями запрос и получает результат в виде строки JSON.

Вторая часть интерфейса пользователя реализована в исполняемом модуле `connector`. Он является связующим звеном для пользовательского интерфейса и приложения и служит для приёма команды в виде строки JSON, её десериализации и непосредственной передачи модулю описания команд. К пользовательскому интерфейсу он относится по той причине, что скрывает сам факт его наличия. Внеся незначительные изменения в этот модуль, можно убрать зависимость от пользовательского интерфейса совсем (например, читая настройки и начальные команды из загрузочного файла). Данный модуль запускается как отдельное приложение и имеет следующую функциональность:

1) `main` – функция, которая в бесконечном цикле ожидает приходящие от пользователя команды, и, когда они пришли, отправляет их на обработку. Если есть результат обработки – отправляет его пользователю.

2) `_process_incoming_command` – данная функция принимает сериализованную в строку JSON команду и ссылку на класс описания команд, десериализует её и отправляет на исполнение, возвращает сериализованный в строку JSON результат выполнения команды (если он есть).

3.2 Модуль описания команд

Данный модуль предназначен для выполнения команд, поступивших от пользователя. Интерфейс этого модуля достаточно прост:

```
class CommandsDescriptionInterface(metaclass=ABCMeta):

    @abstractmethod
    def execute_command(self, command):
        pass
```

В функцию `execute_command` передаётся объект команды, который определён с помощью класса `Command`:

```
class Command:
    def __init__(self, target_block, action,
        additional_information):
        self._target_block = target_block
        self._action = action
        self._additional_information =
```

```

        additional_information

    @property
    def target_block(self):
        return self._target_block

    @property
    def action(self):
        return self._action

    @property
    def additional_information(self):
        return self._additional_information

```

Класс команды имеет следующие свойства:

- `target_block` – идентификатор целевого блока, к интерфейсу которого нужно обратиться для того что бы выполнить команду;
- `action` – идентификатор действия, которое описывает данную команду;
- `additional_information` – дополнительная информация, которая различается в зависимости от того какая команда исполняется в данный момент;

Идентификаторы целевых блоков описаны в виде набора констант в модуле `target_block_types`:

```

SETTINGS_BLOCK           = 0x001
EVENT_LOG_BLOCK          = 0x002

RUSES_MANAGEMENT_BLOCK   = 0x003
MONITOR_INTERACTION_BLOCK = 0x004

```

Идентификаторы действий также описаны в виде набора констант модуля `action_types`:

```

CREATE      = 0x001
READ        = 0x002
UPDATE      = 0x003
DELETE      = 0x004
EXECUTE     = 0x005
RESTORE     = 0x006
EXPORT      = 0x007
IMPORT      = 0x008
START       = 0x009
STOP        = 0x00A

```

Интерфейс `CommandsDescriptionInterface` реализован классом `CommandsDescriptionModule` в модуле `commands_description`. Данный класс содержит в себе таблицу отношений между командой и её

обработчиком, которая фактически описывает API приложения. Ключом данной таблицы является пара идентификаторов: идентификатор целевого блока и идентификатор действия. В обработчике разбираются аргументы команды, и она делегируется модулю, который должен её выполнить. Список обработчиков команд, которые могут быть выполнены приложением выглядит следующим образом:

1) `_create_setting_handler` – обработчик команды задания новой настройки приложения. В качестве входного параметра принимает объект настройки (пара ключ\значение);

2) `_read_settings_handler` – обработчик команды чтения настройки приложения. Возвращаемое значение зависит от входного параметра: если входной параметр не определён – возвращается список всех настроек, если входной параметр является списком ключей – возвращается список настроек по списку, если входной параметр является одиночным ключом – возвращается значение по этому ключу;

3) `_update_setting_handler` – обработчик команды обновления ранее созданной настройки. В качестве входного параметра принимает новый объект настройки;

4) `_delete_setting_handler` – обработчик команды удаления существующей настройки. В качестве входного параметра принимает ключ настройки;

5) `_import_settings_handler` – обработчик команды импорта настроек. В качестве входного параметра принимает путь к внешнему файлу с настройками;

6) `_export_settings_handler` – обработчик команды экспорта настроек. В качестве входного параметра принимает путь к целевому файлу, в который приложение запишет настройки.

7) `_read_event_log_handler` – обработчик команды чтения журнала событий файловой системы. В качестве входного параметра передаётся промежуток дат. Возвращает события, произошедшие в системе за этот промежуток;

8) `_restore_state_by_event_log_handler` – обработчик команды восстановления файловой системы по журналу событий. В качестве входного параметра принимает либо идентификатор события, либо промежуток идентификаторов. Изменения, вызванные событием, либо набором событий по возможности откатываются приложением в исходное состояние;

9) `_create_rule_handler` – обработчик команды создания правила. В качестве входного параметра принимает объект правила;

10) `_read_rules_handler` – обработчик команды чтения всех правил, заданных в приложении. Возвращает коллекцию правил;

11) `_update_rule_handler` – обработчик команды обновления правила. В качестве входного параметра принимает объект правила;

12) `_delete_rule_handler` – обработчик команды удаления правила.

В качестве входного параметра принимает идентификатор правила;

13) `_import_rules_handler` – обработчик команды импорта правил. В качестве входного параметра принимает путь к внешнему файлу с правилами;

14) `_export_rules_handler` – обработчик команды экспорта правил. В качестве входного параметра принимает путь к целевому файлу;

15) `_execute_action_handler` – обработчик команды немедленного выполнения действия над указанным файлом. В качестве входного параметра принимает путь к файлу и объект действия, который будет рассмотрен позднее;

16) `_start_monitor_handler` – обработчик команды запуска монитора;

17) `_stop_monitor_handler` – обработчик команды остановки монитора;

Если обработчик команды не найден по ключу – генерируется исключение `NotImplementedError`, информация об исключении записывается в журнал и приложение продолжает работать в обычном режиме.

Поскольку данный модуль предназначен для выполнения команд, поступивших от пользователя, он непосредственно связан с половиной модулей приложения, среди которых:

- 1) Модуль интерфейса пользователя.
- 2) Модуль журналирования.
- 3) Модуль настройки приложения.
- 4) Модуль управления правилами.
- 5) Модуль взаимодействия с монитором.

3.3 Модуль журналирования

Модуль журналирования предназначен для записи отчётов о всех действиях приложения в файловой системе пользователя в базу данных, а также для записи в файл журнала всех отладочных сообщений и сообщений об ошибках и предупреждениях. Модуль журналирования представлен в приложении классом `LoggerModule`:

```
class LoggerModule(metaclass=Singleton):

    def __init__(self, filename, database):
        pass

    def write_event_info(self, log_record):
        pass

    def read_event_info(self, log_record_id):
        pass
```

```

def read_events_info_range(self, start_record_id,
                           end_record_id):
    pass

def read_events_info_daterange(self, start_date,
                                end_date):
    pass

@property
def logger(self):
    pass

```

Методы `write_event_info` и `read_event_info` предназначены для записи и чтения события в файловой системе соответственно. Методы `read_events_info_range` и `read_events_info_daterange` позволяют читать данные по промежуткам идентификаторов и дат создания записей.

Свойство `logger` предоставляет доступ к объекту класса `Logger` встроенного в язык Python модуля `logging`, который позволяет настроить журналирование строк информации по типам в файл или консоль, настроить формат записей, задать обработчики для разных типов записей и т.д.

Формат записи в файле журнала будет выглядеть следующим образом:

```
'%(levelname)s [% (asctime)s]: %(message)s'
```

Так же стоит отметить что модуль журналирования может быть создан только в одном экземпляре, что описывается метаклассом (классом который управляет созданием объектов другого класса) `Singleton`:

```

class Singleton(type):
    """ Use to create a singleton object.
    """

    def __init__(cls, name, bases, dict):
        super().__init__(name, bases, dict)
        cls._instance = None

    def __call__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(Singleton,
                                   cls).__call__(*args, **kwargs)
        return cls._instance

```

Логика метода `__call__` запускается в момент создания объекта класса, к которому добавлен вышеописанный метакласс и не даёт создать объект `cls._instance` если он уже был ранее создан.

Модуль журналирования используется модулем взаимодействия с

монитором файловой системы для записи действий над файлами и отладочной информации, и модулем описания команд для предоставления пользователю возможности читать информацию из журнала и откатывать по ней действия, произошедшие в файловой системе ранее.

3.4 Модуль настройки приложения

Модуль настройки приложения предоставляет доступ к файлу настроек как к словарию значений по ключам. Так же он предоставляет возможность импортировать и экспортировать файл настроек приложения во время его работы. Модуль настройки представлен классом `SettingsModule`, экземпляр которого, также, как и экземпляр `LoggerModule`, можно создать лишь один на приложение:

```
class SettingsModule(metaclass=Singleton):
    def __init__(self):
        pass

    def __getitem__(self, key):
        pass

    def __setitem__(self, key, value):
        pass

    def __delitem__(self, key):
        pass

    def get_all_settings(self):
        pass

    def get_settings_by_keys(self, keys_list):
        pass

    def export_settings(self, destination_path):
        pass

    def import_settings(self, source_path):
        pass
```

Методы `__getitem__`, `__setitem__`, `__delitem__` добавляют объекту класса функциональность индексатора, то есть дают возможность обращаться к настройкам, инкапсулированным классом, следующим образом:

- `value = obj[key]` – чтение настройки по ключу;
- `obj[key] = value` – запись настройки по ключу;
- `del obj[key]` – удаление настройки по ключу;

Метод `get_all_settings` позволяет получить весь список настроек, что полезно для обеспечения доступа к ним через интерфейс пользователя.

Метод `get_settings_by_keys` принимает на вход список ключей и возвращает настройки по этим ключам, что позволяет на стороне интерфейса пользователя разделить настройки на секции.

Методы `export_settings` и `import_settings` принимают в качестве входных параметров путь к файлу для экспорта и путь к файлу для импорта соответственно. Экспорт настроек побочных эффектов не имеет, но после импорта нужно перезапустить приложение для того чтобы все настройки применились.

Модуль настроек используется модулем описания команд для предоставления пользователю возможности их изменять и инициализации некоторых модулей системы. Также настройки используются модулем взаимодействия с монитором во время работы.

3.5 Модуль хранения данных

Модуль хранения данных предоставляет остальному приложению простые и надёжные интерфейсы по работе с базой данных MongoDB. База MongoDB была выбрана исходя из того, что она обладает высокой скоростью записи и нефиксированной структурой документов. Это позволяет хранить в коллекции документы (BSON объекты) с различающимся набором полей, что очень важно при хранении правил и записей журнала, где данные могут различаться в зависимости от сложности правила или типа произошедшего события.

Поскольку база данных не реляционная, и таблицы со связями в ней отсутствуют, привести схему данных не представляется возможным. Но поскольку в базе данных, используемой описываемым приложением только две коллекции с данными: коллекция информации о действиях, произведенных приложением над файлами и коллекция правил, можно привести пример типовых документов, которые будут в этих коллекциях храниться. Пример документа, хранящего в себе информацию о удалении приложением файла:

```
{
  "_id": ObjectId("12edf42342fgg"),
  "target_file": "/home/username/temp/file.txt",
  "action": {
    "action_type": 1
    "is_permanent_deleting": FALSE
  }
}
```

Как можно увидеть далее, документ, который храниться в базе, полностью совпадает по набору свойств с объектом языка Python, что позволяет заметно упростить разработку и убрать расход времени на сборку объекта по частям, как это было бы в случае работы с реляционными базами

данных. Документ считывается из базы данных и напрямую отображается в объект Python.

Пример документа из коллекции правил:

```
{
  "_id": ObjectId("12edf42342fgg"),
  "target_directory": "/home/username/temp/",
  "file_constraints": {
    "is_directory": FALSE,
    "target_event_types": [1, 2],
    "target_content_types": [1, 2],
    "target_name_template": Null,
    "target_extention_template": Null,
    "target_file_max_size": Null
    "target_file_min_size": Null
  },
  "action": {
    "action_type": 1,
    "is_permanent_deleting": FALSE
  }
}
```

Интерфейс базы данных предоставляет стандартные CRUD-операции, и объявлен в виде класса `EntityStorageInterface`:

```
class EntityStorageInterface(metaclass=ABCMeta):

    @abstractmethod
    def create(entity):
        pass

    @abstractmethod
    def read(entity_id):
        pass

    @abstractmethod
    def update(entity):
        pass

    @abstractmethod
    def delete(entity_id):
        pass

    @abstractmethod
    def read_all():
        pass
```

Этот интерфейс имеет 2 реализации: `EntityStorage` и `CachedEntityStorage`. Отличаются они тем, что в реализации

`CachedEntityStorage` результат вызова методов `read` и `read_all` кэшируются. Методы `create`, `update`, `delete` работают так же, как и аналогичные методы `EntityStorage` с той лишь разницей, что перед тем как создать изменить или удалить сущность в базе данных они должны сделать изменения в массиве закэшированных объектов. Реализация интерфейса хранения с кэшем нужна для модуля управления правилами, потому что при большом потоке событий от файловой системы неприемлемо каждый раз читать правила из базы данных. Диаграмму классов модуля хранения данных можно увидеть на рисунке 3.1.

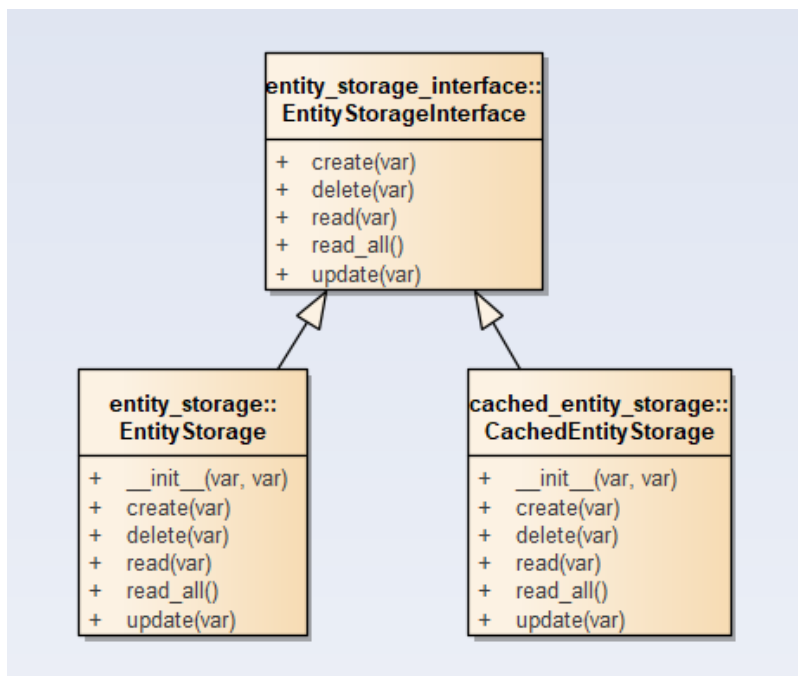


Рисунок 3.1 – Диаграмма классов модуля хранения данных

Каждый экземпляр `EntityStorage` или `CachedEntityStorage` инициализируется коллекцией `MongoDB`, которая является хранилищем документов определённого формата. Для того чтобы получить коллекцию используется объект клиента базы данных, который можно создать следующим образом:

```
client = MongoClient('mongodb://localhost:27017/')
```

Для инициализации клиента `MongoDB` используется строка подключения, представляющая собой адрес базы данных на локальной машине.

Далее с помощью объекта клиента по имени извлекается нужная база данных, через которую в свою очередь есть доступ ко всем коллекциям. После того как коллекция с нужным именем получена, доступ ко всем документам, находящимся в коллекции, можно получить используя API базы данных. API коллекций `MongoDB` достаточно для реализации всех методов интерфейса.

3.6 Модуль анализа файлов

Модуль анализа файлов предназначен для сбора всей статистики по файлу, для которого произошло событие в файловой системе. Модуль используется только модулем взаимодействия с монитором файловой системы и представлен классом `FileAnalyzer`, имеющим следующий интерфейс:

```
class FileAnalyzer:
    def __init__():
        pass

    @staticmethod
    def analyse_file(file_path):
        pass
```

Интерфейс класса содержит единственный метод, который принимает путь к файлу и возвращает объект с информацией о нём. Объект этот представлен в приложении классом `AnalysisResult`:

```
class AnalysisResult:
    def __init__(self):
        self._is_directory = False
        self._name = None
        self._extension = None
        self._content_type = None
        self._size = None
        self._content_specific_info = None

    @property
    def is_directory(self):
        return self._is_directory

    @is_directory.setter
    def is_directory(self, value):
        self._is_directory = value

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def extension(self):
        return self._extension

    @extension.setter
```

```

def extension(self, value):
    self._extension = value

@property
def content_type(self):
    return self._content_type

@content_type.setter
def content_type(self, value):
    self._content_type = value

@property
def size(self):
    return self._size

@size.setter
def size(self, value):
    self._size = value

@property
def content_specific_info(self):
    return self._content_specific_info
@content_specific_info.setter
def content_specific_info(self, value):
    self._content_specific_info = value

```

Данный класс содержит следующие свойства:

- `is_directory` – определяет директория это или обычный файл;
- `name` – имя файла либо директории;
- `extension` – расширение файла, имеет значение `None` в случае директории;
- `content_type` – идентификатор типа содержимого файла;
- `size` – размер файла
- `content_specific_info` – информация, специфичная для определённого типа содержимого;

Стоит отдельно обратить внимания на два свойства: `content_type` на и `content_specific_info`. Наличие этих двух свойств в результате анализа файла позволят пользователю задавать правила, касающиеся не только файлов с определённым расширением или именем, но файлов с определённым с определённым типом контента. Типы контента доступные для задания в правилах и обнаруживаемые модулем анализа файлов описаны в виде набора констант в модуле `content_types`:

ARCHIVE	= 0x001
NOTE	= 0x002
DOCUMENT	= 0x003
BOOK	= 0x004

AUDIO	= 0x005
VIDEO	= 0x006
BINARY	= 0x007
SYSTEM	= 0x008
IMAGE	= 0x009

Информация, предоставляемая свойством `content_specific_info` зависит от типа контента, что позволяет пользователю добавлять специфические правила, и описывается следующей иерархией классов:

1) `ContentSpecificInfo` – класс, являющийся общим предком всех классов для специфичной по типу контента информации. Содержит идентификатор типа контента, описанный в модуле `content_types`.

2) `MusicSpecificInfo` – наследник `ContentSpecificInfo`, содержит в себе свойства, позволяющие получить информацию о музыкальных файлах: название группы, жанр музыки и название альбома.

3) `TextSpecificInfo` – наследник `ContentSpecificInfo`, содержит в себе свойства, предоставляющие информацию о теме и языке текстового документа.

4) `DocumentSpecificInfo` – наследник `TextSpecificInfo`, не содержит дополнительной информации относительно базового класса.

5) `BookSpecificInfo` – наследник `TextSpecificInfo`, содержит свойства, специфичные для книг и позволяющие кроме темы и языка получить информацию о авторе и жанре книги.

Диаграмма классов для вышеописанной иерархии приведена на рисунке 3.2.

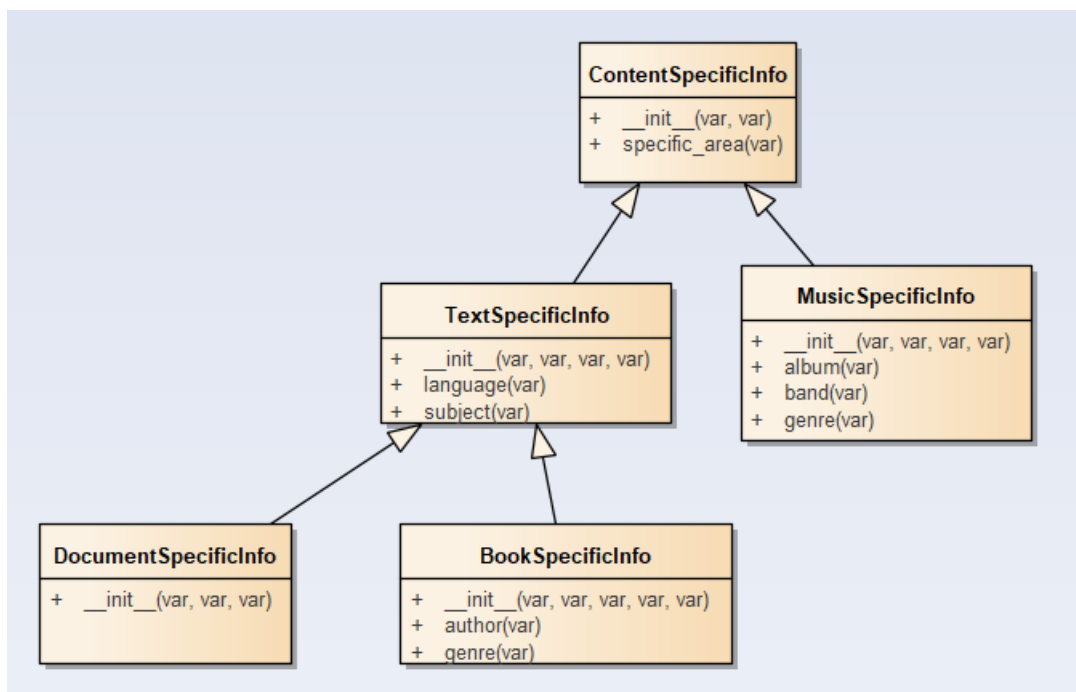


Рисунок 3.2 – Диаграмма классов для информации, зависящей от назначения содержимого файлов

3.7 Модуль пересылки сообщений

Приложение для данного дипломного проекта состоит из трёх отдельных программ: программа мониторинга файловой системы, программа анализа и распределения файлов, программа-клиент. Модуль пересылки сообщений отвечает за обмен данными между этими тремя программами.

Данные передаются в виде текстовых строк, в формате, который заранее описан протоколом между отправителем и получателем. В качестве инструмента для передачи сообщений используется библиотека ZeroMQ, предоставляющая надёжный интерфейс для обмена сообщениями через сокеты и позволяющая быстро построить различные шаблоны взаимодействия, такие как точка-точка, издатель-подписчик и т.д.

Модуль пересылки сообщений имеет следующий интерфейс:

```
class MessagingInterface(metaclass=ABCMeta):

    @abstractmethod
    def send_message(self, message):
        pass

    @abstractmethod
    def receive_message(self):
        pass
```

Реализует интерфейс обмена сообщениями класс `MessagingManager`, который описан следующим образом:

```
class MessagingManager(MessagingInterface):
    def __init__(self, messaging_manager_type, \
        socket_binding_address):
        self._send_queue = Queue()
        self._receive_queue = Queue()
        self._background_worker_stop_event = Event()
        self._background_worker = None

    def __del__(self):
        pass

    def send_message(self, message):
        pass

    def is_received_messages_exists(self):
        pass

    def receive_message(self):
        pass

    def get_all_received_messages(self):
```

pass

При инициализации объекта класса в метод конструктора `__init__` передаётся тип менеджера пересылки сообщений – клиент или сервер, и адрес, к которому будет привязан сокет для обмена сообщениями. Поэтому для того что бы создать соединение, каждая из программ, участвующих в обмене, должна создать объект своего типа (зависит от назначения программы) и передать в качестве параметра конструктора один и тот же адрес. Создание пары таких объектов выглядит следующим образом:

```
server = MessagingManager(MessagingManagerType.SERVER,
                           "tcp://127.0.0.1:5555")
client = MessagingManager(MessagingManagerType.CLIENT,
                           "tcp://127.0.0.1:5555")
```

Также при инициализации создаются две очереди класса `Queue` для принятых и переданных сообщений, объект события `_background_worker_stop_event` класса `Event`, а также объект класса `MessagingManagerBackgroudReciever`, который представляет собой поток `Thread` основной задачей которого является приём приходящих и отправка исходящих сообщений в бесконечном цикле. Объект события нужен для того, чтобы при удалении сборщиком мусора объекта менеджера пересылки сообщений (и соответственно вызова деструктора `__del__`), можно было остановить поток приёма-передачи сообщений.

Передача данных между двумя потоками идёт через потокобезопасные очереди `_send_queue` и `_receive_queue`. При вызове метода `send_message`, в который в качестве параметра передаётся строка сообщения, объект класса `MessagingManager` не отправляет сообщение не передаёт сообщение непосредственно, а кладёт его в очередь `_send_queue`. Методы `receive_message` и `get_all_received_messages` возвращают сообщения, читая их из очереди `_receive_queue`. Таким образом, использующая эту функциональность программа получает асинхронную реализацию приёма\передачи сообщений.

Метод `is_received_messages_exists` предназначен для проверки очереди на наличие принятых сообщений. Он используется в тех случаях, когда код, принимающий сообщения в бесконечном цикле, ждёт их поступления.

Класс `MessagingManagerBackgroudReciever` описан следующим образом:

```
class MessagingManagerBackgroudReciever(Thread):
    def __init__(self, messaging_manager_type, \
                 socket_binding_address, queues,
                 events):
        super().__init__()
        self._send_queue, self._receive_queue = None
```

```

        stop_event = None
        self._send_queue = None
        self._recieve_queue = None
        self._stop_event = None
        self._context = None
        self._socket = None
    def _bind_socket_to_address(self, \
                                messaging_manager_type,
                                binding_address):
        pass

    def run(self):
        pass

```

При создании объекта этого класса создаётся контекст ZeroMQ, с помощью этого контекста создаётся сокет, предназначенный для пересылки сообщения. Код создания сокета следующий:

```

self._socket = self._context.socket(zmq.PAIR)

```

Константа `zmq.PAIR` определяет для сокета шаблон взаимодействия точка-точка - это значит, что участников передачи может быть максимум двое и они равноправны. Но несмотря на то что участники передачи равноправны, интерфейс сокетов ZeroMQ спроектирован так что один из сокетов должен быть привязан к адресу с помощью метода `bind` и являться условным сервером, а второй с помощью метода `connect` и являться условным клиентом. Разделяются они с помощью параметра конструктора `messaging_manager_type` который определяется как перечисление:

```

class MessagingManagerType(Enum):
    CLIENT = 1
    SERVER = 2

```

За привязку сокета к адресу отвечает метод `_bind_socket_to_address`, который вызывает методы сокета `bind` или `connect` в зависимости от типа менеджера пересылки сообщений.

Метод `run` запускается при старте потока и обслуживает очереди входящих и исходящих сообщений в цикле. Цикл завершается, когда объект события `_stop_event` оказывается в активном состоянии. После того как цикл завершился объект сокета закрывается, а объект контекста уничтожается.

3.8 Модуль управления правилами.

Модуль управления правилами является одним из ключевых модулей приложения и отвечает за всю логику работы с правилами распределения файлов. Он связан с модулем описания команд, через который эти правила

задаются, изменяются и удаляются и модулем взаимодействия с монитором файловой системы, который для каждого события получает список правил и применяет их.

Правило распределения файлов, в свою очередь является ключевой сущностью и представлено классом `Rule`:

```
class Rule:

    def __init__(self, target_directory,
file_constraints,\
        action):
        self._target_directory = target_directory
        self._file_constraints = file_constraints
        self._action = action

    @property
    def target_directory(self):
        return self._target_directory

    @property
    def file_constraints(self):
        return self._file_constraints

    @property
    def action(self):
        return self._action
```

Класс правила представлен всего тремя свойствами:

1) Свойство `target_directory` представляет собой путь к директории и позволяет определить, к файлам какой отслеживаемой директории это правило относится.

2) Свойство `file_constraints` задаёт список ограничений, которым должен соответствовать файл что бы к нему было применено это правило. Значение свойства представлено классом `RuleFileConstraint`.

3) Свойство `action` определяет действие, которое применяется к файлу, удовлетворяющему условию этого правила. Значение свойства представлено классом-наследником класса `RuleAction`.

Определение класса, представляющего информацию о ограничениях целевого файла - `RuleFileConstraint`:

```
class RuleFileConstraint:

    def __init__(self):
        self._is_directory = False
        self._target_event_types = []
        self._target_content_types = []
        self._target_name_template = None
```

```
self._target_extension_template = None
self._target_file_max_size = None
self._target_file_min_size = None
```

Список ограничений, которые могут быть определены пользователем и применены к файлу:

1) `is_directory` – если значение свойства истинно – правило применяется только к директориям. Если ложно – и к директориям, и к файлам.

2) `target_event_types` – свойство представляющее собой массив событий. Если событие, произошедшее с файлом, не входит в этот массив – правило к нему не применяется.

3) `target_content_types` – свойство представляющее собой массив типов контента. Если тип контента файла не подходит – правило не применяется.

4) `target_name_template` – представляет собой шаблон регулярного выражения. Имя файла должно совпадать с этим шаблоном.

5) `target_extension_template` – также шаблон регулярного выражения. Расширение файла должно подпадать под этот шаблон.

6) `target_file_max_size` – ограничение по максимальному размеру файла.

7) `target_file_min_size` – ограничение по минимальному размеру файла.

Не обязательно определять все из вышеперечисленных ограничений при создании правила, но всем, что были определены, информация о файле и информация о событии, произошедшем с файлом, должны соответствовать.

Информация о действии, представленная свойством `action` представлена иерархией классов, каждый класс в которой описывает конкретное действие и свойства специфичные для выполнения этого из действия. Иерархия классов может быть описана следующим образом:

1) `RuleAction` – базовый класс для всех действий в правиле. Содержит идентификатор действия.

2) `DeleteFileRuleAction` – класс, представляющий удаление файла. Содержит свойство, предоставляющее информацию о том, нужно ли удалять файл навсегда или его стоит переместить в корзину.

3) `ReplaceFileRuleAction` – класс, представляющий перемещение файла. Поскольку файл может быть перемещён как в уже существующую директорию, так и в директорию, которая может быть создана на основе его имени – класс содержит свойства, задающие путь к целевой директории либо шаблон имени целевой директории.

4) `RenameFileRuleAction` – класс, представляющий переименование файла. Содержит свойство, задающее шаблон нового имени файла.

5) `GroupByAttributeFileRuleAction` – класс, описывающий действие по группировке файлов по определённому признаку. Признак

задаётся типом действия. Содержит в себе свойство, задающее шаблон целевой директории.

После того как действие, заданное свойством action правила распределения, применится к файлу или директории, данные о файле и действии записываются в базу данных для возможности восстановления. Диаграмма классов иерархии действий представлена на рисунке 3.3.

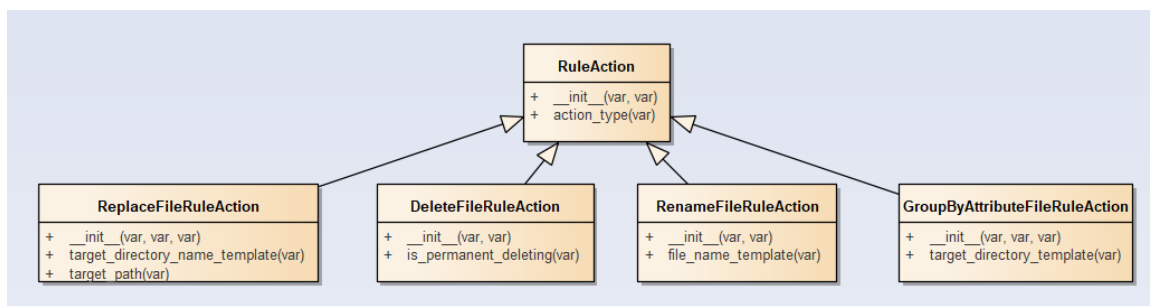


Рисунок 3.3 – Диаграмма классов для иерархии действий правила

Идентификатор действия определён в виде набора констант в модуле rule_action_types:

```

# common actions
DELETE_FILE = 0x001
REPLACE_FILE = 0x002
RENAME_FILE = 0x003
IGNORE = 0x00B
# music actions
GROUP_BY_MUSIC_BAND = 0x004
GROUP_BY_MUSIC_GENRE = 0x005
GROUP_BY_MUSIC_ALBUM = 0x006
# books and docs actions
GROUP_BY_SUBJECT = 0x007
GROUP_BY_LANGUAGE = 0x008
GROUP_BY_GENRE = 0x009
GROUP_BY_AUTHOR = 0x00A
  
```

Модуль, позволяющий работать с вышеописанными правилами, реализован в виде класса RulesManager. Данный класс предоставляет глобальный интерфейс для работы с правилами для модуля описания команд и модуля взаимодействия с монитором файловой системы:

```

class RulesManager(metaclass=Singleton):

    def __init__(self):
        pass

    def create_rule(self, rule):
        pass
  
```

```

def update_rule(self, rule):
    pass
def delete_rule(self, rule_id):
    pass
def read_all_rules(self):
    pass
def get_rules(self, event_info,
               analyze_results=None):
    pass

def export_rules(self, target_file):
    pass

def import_rules(self, source_file):
    pass

def add_directory_list_change_handlers(self,
                                       add_callback, remove_callback):
    pass

```

Методы `create_rule`, `update_rule`, `delete_rule` и `read_all_rules` делегируют вызовы объекту класса `CachedEntityStorage`, напрямую передавая им свои входные параметры. Кроме того, в методах `create/update/delete_rule` отслеживается появление новых или полное удаление старых отслеживаемых директорий.

Метод `get_rules` принимает информацию о событии в файловой системе и опциональный результат анализа файла и возвращает все правила, соответствующие этому событию.

Методы `export_rules` и `import_rules` предназначены для экспорта и импорта правил в систему соответственно, пути к необходимым для этого файлам передаются в качестве параметра метода. Данные импортируются и экспортируются в формате JSON.

Метод `add_directory_list_change_handlers` позволяет добавить два обработчика, один из которых сработает если список директорий, которые отслеживаются приложением – увеличился, другой – если он уменьшился. К примеру: пользователь добавил правило для новой директории или удалил правило, которое является последним для определённой директории. И в том и в другом случае программа-монитор должна быть оповещена, чтобы обновить список отслеживаемых директорий и не генерировать бесполезные события.

3.9 Модуль взаимодействия с монитором

Модуль взаимодействия с монитором является центральным модулем приложения. Его главной задачей является обработка сообщений, приходящих от монитора файловой системы и передача сообщений от других модулей, в

частности от модуля управления правилами и от модуля описания команд, монитору. Реализация модуля представлена двумя классами: MonitorMessagesProcessingLoop и MonitorInteractionManager:

```
class MonitorMessagesProcessingLoop(Thread):

    def __init__(self, messaging_manager, stop_event):
        pass

    def _process_messages(self, messages):
        pass

    def run(self):
        pass


class MonitorInteractionManager(metaclass=Singleton):

    def __init__(self):
        self._send_queue = None
        self._receive_queue = None
        self._logger_module = None
        self._settings_module = None
        self._messaging_manager = None
        self._rules_manager = None
        self._stop_event = None
        self._processing_loop = None

    def __del__():
        pass

    def _add_watching_directory_handler(self,
        directory_path):
        pass

    def _remove_watching_direcotory_handler(self,
        directory_path):
        pass

    def rollback_actions(self, event_log_records):
        pass

    def execute_action_immediately(self, file_path,
        action):
        pass

    def start_monitor(self):
        pass

    def stop_monitor(self):
        pass
```

Класс `MonitorMessagesProcessingLoop` представляет собой объект потока, который в бесконечном цикле в методе `run` обменивается сообщениями с программой монитором. Метод `_process_messages` вызывается из метода `run` и создаёт новый поток обработки для каждого из пришедших от монитора сообщений.

Класс `MonitorInteractionManager` агрегирует в себе реализации большинства ранее описанных модулей, а также содержит две потокобезопасные очереди для обмена сообщениями и объект потока обработки `_processing_loop` и событие для остановки этого потока в случае удаления менеджера - `_stop_event`, которое устанавливается в деструкторе `__del__`.

Метод `_add_watching_directory_handler` служит как функция-обработчик, которая устанавливается через модуль управления правилами и срабатывает при добавлении новой отслеживаемой директории. То же самое, но для удаления директории — метод `_remove_watching_directory_handler`.

Метод `execute_action_immediately` позволяет выполнить заданное пользователем действия, не смотря ни на какие правила немедленно. Этот метод полезен в случае если пользователю придётся вручную откатывать последствия некоторых операций. Что бы какое-то из заданных пользователем правил, действие которого было для него неприемлемым не применилось снова, перед тем как исполнить над файлом заданное пользователем действие для него добавляется правило с действием `IGNORE`, которое отменяет все остальные.

Методы `start_monitor` и `stop_monitor` предназначены для передачи команды монитору о старте и остановке соответственно. Это может быть полезно в ситуациях, когда пользователя устраивает работа приложения, но ему нужно скопировать большое число файлов в отслеживаемую директорию, и он знает, что работа монитора скажется на производительность персонального компьютера. В таком случае монитор останавливается пользователем в целях оптимизации и запускается после завершения действия.

3.10 Модуль мониторинга файловой системы

Данный модуль предназначен для отслеживания и передачи сообщений о событиях, происходящих в отслеживаемых директориях. Функциональность монитора реализована в виде исполняемого модуля `monitor`. Монитор имеет следующий набор функций:

1) `main` — главная функция модуля, в которой создаётся контекст `Inotify` (библиотеки, отвечающей за отслеживание событий), и запускается цикл, в котором по порядку обслуживаются сначала приходящие сообщения, затем события от файловой системы.

2) `process_recieved_messages` — функция, предназначенная для

приёма и обработки принятых сообщений. Принимает в качестве параметров контекст Inotify, и список сообщений. С помощью контекста удаляются и добавляются отслеживаемые директории.

3) `process_recieved_events` – функция, принимающая в качестве параметров объект контекста и менеджер пересылки сообщений. Считывает несколько сообщений из буфера контекста, конвертирует их в сообщения и отправляет модулю взаимодействия с монитором.

4) `process_possible_pair_event` - функция, которая обрабатывает ситуацию, когда событие может быть парным. Например, переименование файла это два события, потому что это фактически перемещение файла в файл с другим именем. Функция принимает на вход текущее событие и список с возможно-парными событиями.

5) `get_message_by_event_pair` – функция, которая принимает парное событие и возвращает объект сообщения для этого события.

6) `get_message_by_event` – функция, которая принимает событие и возвращает объект сообщения для этого события.

7) `get_event_type_by_mask` – функция, позволяющая перевести флаг события Inotify во внутренний для приложения тип события.

Тип события от монитора в приложении представлен набором констант в модуле `monitor_event_types`:

<code>FILE_CREATED</code>	<code>= 0x001</code>
<code>FILE_NAME_CHANGED</code>	<code>= 0x002</code>
<code>FILE_CONTENT_CHANGED</code>	<code>= 0x004</code>
<code>FILE_INCLUDED</code>	<code>= 0x008</code>
<code>FILE_EXCLUDED</code>	<code>= 0x010</code>
<code>FILE_DELETED</code>	<code>= 0x020</code>
<code>DIRECTORY_REPLACED</code>	<code>= 0x040</code>
<code>DIRECTORY_DELETED</code>	<code>= 0x080</code>
<code>FILE_METADATA_CHANGED</code>	<code>= 0x100</code>

Объект сообщения, который передаётся от монитора и к монитору описан классом `MonitorMessage`:

```
class MonitorMessage:

    def __init__(self):
        self._target_directory = ""
        self._target_file = ""
        self._additional_information = ""
        self._event_type = 0

    def __repr__(self):
        return self.__combine_message()

    def __str__(self):
        return self.__combine_message()
```

```
def __combine_message(self):
    pass

    @staticmethod
    def restore_from_string(message):
        pass
```

Сообщение состоит из следующих частей:

- `target_directory` – имя отслеживаемой директории;
- `target_file` – имя файла, с которым случилось событие;
- `additional_information` – дополнительная информация, например, новое имя файла при переименовании;
- `event_type` – тип события от монитора, представляет собой одну из констант, описанных выше;

Метод `__combine_message` позволяет преобразовать объект в строку, где значения свойств разделяются двоеточиями. В этом формате сообщение пересылается модулю взаимодействия с монитором. Методы `__repr__` и `__str__` позволяют привести объект к строковому формату, что позволяет распечатывать его функцией `print` и переводить в строку функцией `str`. Метод `restore_from_string` используется на принимающей стороне для восстановления сообщения в объектный вид.

В данном разделе были проанализированы все модули из которых состоит приложение данного дипломного проекта, описаны все функции, классы и списки констант, детально описано взаимодействие между модулями и форматы передачи данных. Детальную диаграмму классов приложения можно увидеть на (ссылка на и диаграмму классов).

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Алгоритм работы функции process_received_events

Функция `process_received_events` предназначена для обработки и отправки событий файловой системы в отслеживаемых директориях. Она принимает в качестве аргументов объект контекста `Inotify` – `monitor`, менеджер обмена сообщениями – `monitor_messaging_manager`, список, предназначенный для возможных парных событий – `pair_events_list`, и объект события `Event` – `stop_events_processing_flag`. Сама функция выглядит следующим образом:

```
def process_received_events(monitor, \
    messaging_manager, pair_events_list, \
    stop_events_processing_flag):

    for event in monitor.event_gen():
        if event is not None:
            messages = process_possible_pair_event(event, \
                pair_events_list)

            if messages is None:
                message = get_message_by_event(event)

                messaging_manager.send_message(str(message))
            else:
                for message in messages:
                    messaging_manager.send_message( \
                        str(message))
        else:
            if stop_events_processing_flag.is_set():
                stop_events_processing_flag.clear()
                break
```

Основная работа функции происходит в цикле считывания событий из объекта контекста `Inotify`:

```
for event in monitor.event_gen():
```

Метод `event_gen` возвращает объект события если оно есть в очереди событий и `None` если очередь пуста. После цикла идёт проверка того появилось новое событие или нет. Если оно появилось – мы передаём событие и список возможных парных событий функции `process_possible_pair_event`. Эта функция проверяет и формирует сообщения с учётом возможных парных событий файловой системы. Событие считается парным, если оно представляет собой два события на уровне

файловой системы и одно событие на уровне приложения. В качестве примера можно привести переименование файла, которое на уровне файловой системы представлено как перемещение файла со старым именем из директории и включение файла с новым именем в директорию, но на логическом уровне является одним событием. Функция возвращает:

- 1) Пустой список, если текущее событие первое из парных.
- 2) Один объект сообщения если в списке уже было парное событие, а текущее событие также парное
- 3) Два объекта сообщения, если текущее событие не парное, а в списке уже было возможно парное событие.
- 4) `None`, если список пуст и текущее событие не парное.

Далее проверяется вернула ли функция обработки возможных парных событий какие-либо сообщения:

```
if messages is None:
```

Если сообщения\сообщений нет – сообщение получается по событию с помощью функции `get_message_by_event` и отправляется в виде строки с помощью метода `send_message` объекта менеджера пересылки сообщений:

```
message = get_message_by_event(event)
messaging_manager.send_message(str(message))
```

Если сообщения есть – они также отправляются через объект менеджера пересылки сообщений, но уже в цикле:

```
for message in messages:
    messaging_manager.send_message(str(message))
```

В ситуации, когда событие не пришло и метод `event_gen` возвратил `None` проверяется флаг выхода из функции `stop_events_processing_flag`, который устанавливается таймером раз в 10 секунд в вызывающем коде.

```
stop_event_timer = Timer(10, stop_event.set)
stop_event_timer.start()
```

Этот флаг нужен для того что бы обработка событий не занимала всё время монитора и каждые 10 секунд монитор имел бы возможность проверять входящие сообщения. Такое неравное распределение времени объясняется тем, что входящие сообщения присылаются гораздо реже чем отсылаются исходящие.

Блок-схема одного цикла данного алгоритма представлена на рисунке 4.1.

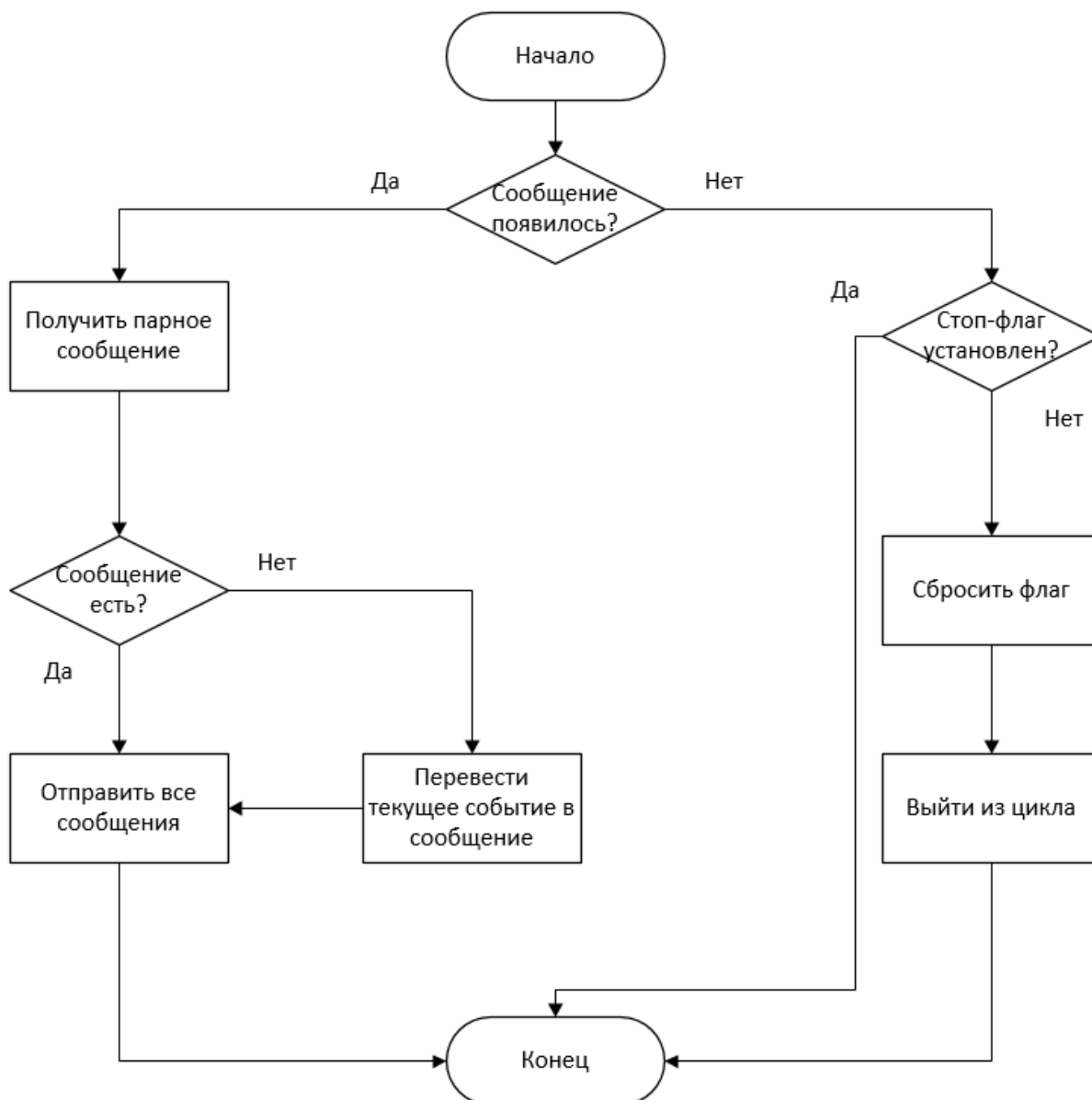


Рисунок 4.1 – Блок схема одного цикла функции `process_received_events`

4.2 Алгоритм работы функции `process_possible_pair_event`

Функция `process_possible_pair_event` предназначена для обработки парных событий, для которых нужно генерировать одно сообщение вместо двух. Примером такого события может служить переименование файла, которое состоит из пары событий: файл перемещён из директории и файл перемещён в директорию. Эту ситуацию надо отличать от событий реального перемещения файла. Код данной функции представлен ниже:

```
def process_possible_pair_event( \
    current_event, pair_event_list):
    (header, _, watch_path, _) = current_event
    if len(pair_event_list) == 0 and is_first_pair_event(
        header):
```

```

        pair_event_list.append(current_event)
        return []
    elif len(pair_event_list) != 0:
        previous_event = pair_event_list.pop()
        (_, _, previous_watch_path, _) = previous_event
        if previous_watch_path == watch_path and \
            is_second_pair_event(header):
            pair_events_message =
                get_message_by_event_pair(
                    previous_event, current_event)
            return [ pair_events_message ]
        elif is_first_pair_event(header):
            pair_event_list.append(current_event)
            previous_event_message =
                get_message_by_event( \
                    previous_event)
            return [ previous_event_message ]
        else:
            previous_event_message =
                get_message_by_event( \
                    previous_event)
            current_event_message = get_message_by_event(
                current_event)
            return [ previous_event_message, \
                current_event_message ]
    return None

```

Функция принимает в качестве аргументов текущее событие `current_event` и список с возможными парными событиями - `pair_event_list`.

Объект `current_event` представляет собой кортеж, из которого нужно выделить заголовок с метаданными события и путь к отслеживаемой директории.

```

(header, _, watch_path, _) = current_event

```

Далее проверяется список возможных парных событий. Если список пуст, нужно проверить может ли текущее событие являться первым из парных. Проверку осуществляет функция `is_first_pair_event`, в которую передаётся заголовок с метаданными о событии. Если оба условия соблюдаются – событие добавляется в список `pair_event_list` и функция возвращает пустой список.

```

pair_event_list.append(current_event)
return []

```

Если в списке уже есть первое парное событие, то оно извлекается и раскладывается на составные части для получения пути к директории:

```
previous_event = pair_event_list.pop()
(_, _, previous_watch_path, _) = previous_event
```

Возникает три ответвления для трёх разных ситуаций. Для первой ситуации проверяется совпадает ли у двух событий имя отслеживаемой директории и, с помощью функции `is_second_pair_event`, является ли текущее событие вторым парным. В случае если оба условия соблюдены – с помощью функции `get_message_by_event_pair` получаем один объект сообщения для двух событий и возвращаем его в виде списка.

```
pair_events_message = get_message_by_event_pair(\
    previous_event, current_event)
return [ pair_events_message ]
```

Для второй ситуации с помощью функции `is_first_pair_event` идёт проверка - является ли текущее событие первым из парных. Если условие соблюдается – текущее событие добавляется в список, а предыдущее событие конвертируется в объект сообщения с помощью функции `get_message_by_event` и возвращается в виде списка.

Если первые две ветки условного оператора не сработали – значит текущее и предыдущее события непарные. В таком случае они оба конвертируются в отдельные сообщения, которые возвращаются в виде списка:

```
previous_event_message
= get_message_by_event(previous_event)
current_event_message =
get_message_by_event(current_event)
return [ previous_event_message, current_event_message ]
```

Если список пуст и текущее событие не является первым парным – возвращается значение `None`, что даёт понять вызывающему коду что текущее событие нужно обрабатывать обычным образом.

В итоге, результатом выполнения функции при таком алгоритме является либо список, который может быть пустым или содержать объекты сообщений либо значение `None`.

Блок-схема алгоритма представлена на рисунке 4.2.

4.3 Алгоритм работы метода `run` потока приёма и передачи сообщений

Метод `run` класса `MessagingManagerBackgroundReceiver`, представляющего собой поток `Thread`, предназначен для последовательного приёма и передачи сообщений. Этот метод запускается при старте потока и при его завершении поток также считается завершившим свою работу. Код метода выглядит следующим образом:

```

def run(self):
    polling_delay = 100

    while not self._stop_event.is_set():

        if self._send_queue.qsize() != 0:
            self._socket.send_unicode( \
                self._send_queue.get())

        events = self._socket.poll(polling_delay)
        if events != 0:
            self._receive_queue.put( \
                self._socket.recv(flags=zmq.NOBLOCK))

    self._socket.close()
    self._context.destroy()

```

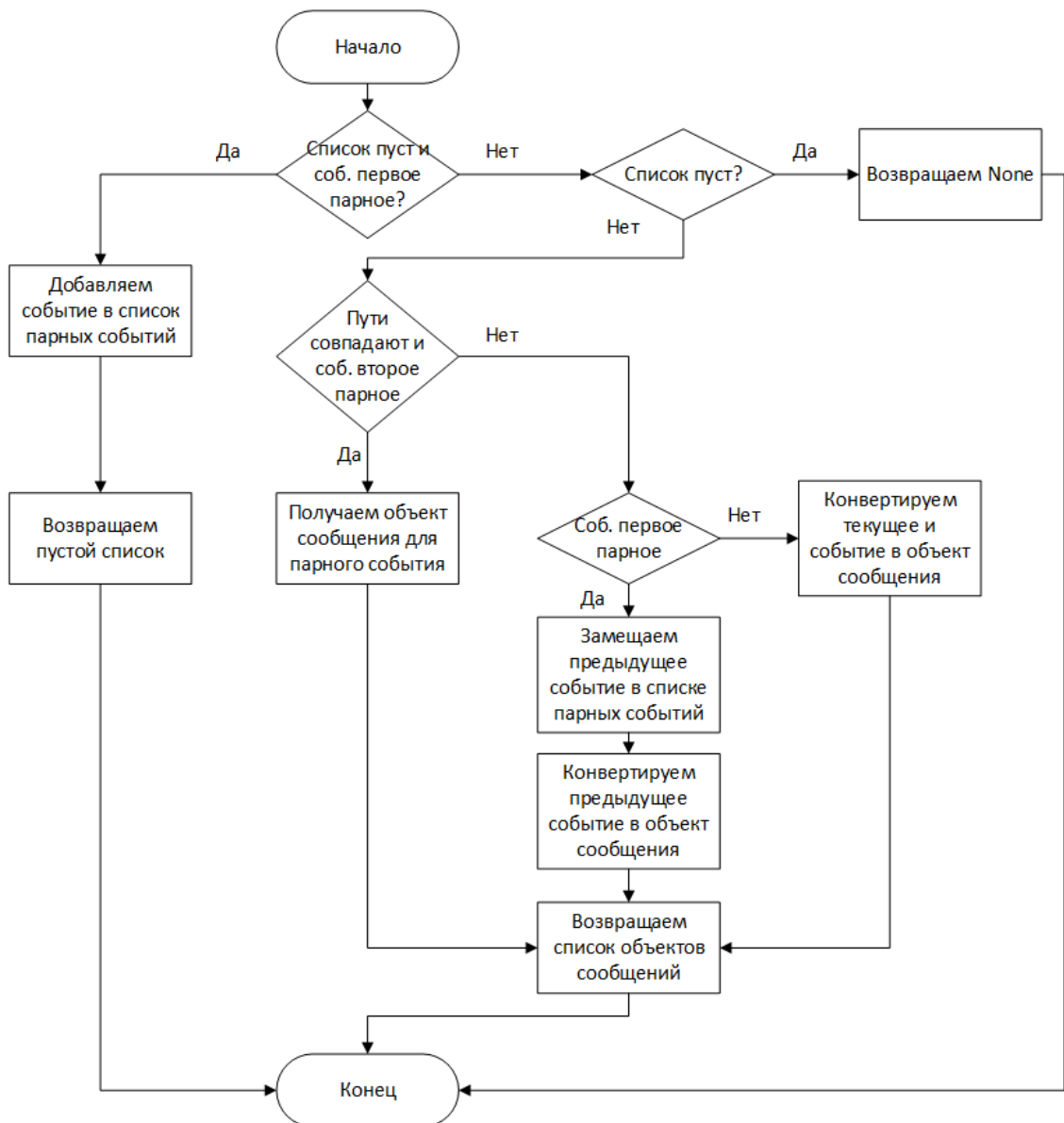


Рисунок 4.2 – Блок-схема алгоритма функции `process_possible_pair_event`

При старте этого метода запускается бесконечный цикл, который работает до тех пор, пока внешний код, создавший данный поток не установит событие, которое позволит выйти из цикла и корректно освободить ресурсы.

На первом этапе цикла проверяется есть ли в очереди `_send_queue` сообщения на отправку. Если очередь не пуста одно сообщение из очереди отправляется с помощью метода `send_unicode` сокета `_socket`. Если очередь пуста – цикл переходит к следующему этапу.

```
if self._send_queue.qsize() != 0
    self._socket.send_unicode( \
        self._send_queue.get())
```

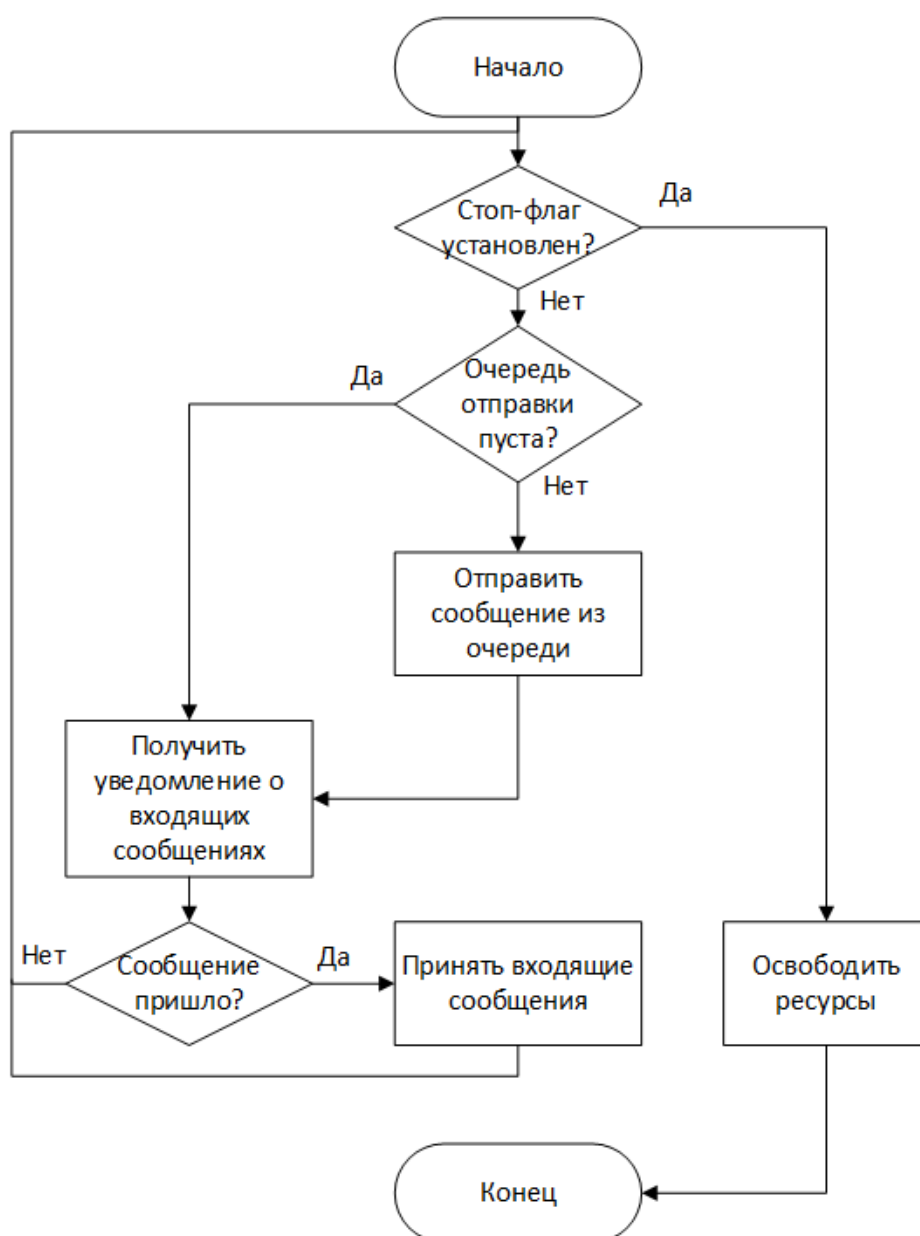


Рисунок 4.3 – Блок-схема алгоритма метода run

На втором этапе идёт попытка получить сигнал о пришедшем

сообщении за промежуток времени 100 мс. Такую возможность предоставляет метод сокета `poll`, куда в качестве задержки передаётся константа `polling_delay` и который возвращает 0, если за заданное время никаких сообщений не пришло. Если сообщение пришло – оно принимается методом сокета `recv` и помещается в очередь принятых сообщений `_receive_queue`.

После того как вызывающий код выставил событие `_stop_event` и цикл завершился, освобождаются все ресурсы потока, а именно: контекст и сокет ZeroMQ.

Блок-схема алгоритма представлена на рисунке 4.3.

4.4 Алгоритм работы метода `get_rules`

Метод `get_rules` класса `RulesManager` предназначен для получения списка правил, которые должны последовательно примениться к файлу, для которого произошло событие в файловой системе. Метод принимает в качестве входных параметров информацию о событии `event_info` и результат анализа файла `analyze_results`. Код метода выглядит следующим образом:

```
def get_rules(self, \
    event_info, analyze_results=None):
    all_suitable_rules = self._get_all_suitable_rules(
        event_info, analyze_results)

    if self._is_ignore_rule_exist(all_suitable_rules):
        return []

    specific_sorted_rules = \

        self._sort_by_specific_level(all_suitable_rules)
    top_specific_rules = \
        self._get_top_specific_rules( \
            specific_sorted_rules)
    delete_rules = self._get_rules_by_action_type( \
        top_specific_rules, action_types.DELETE_FILE)

    if len(delete_rules) != 0:
        return [ delete_rules[0] ]
    return self._get_final_rules_sequence( \
        specific_sorted_rules)
```

Вся информация о событии и файле передаётся в приватный метод класса `_get_all_suitable_rules`, который возвращает список кортежей, где первый элемент это правило, подходящее для данного события, а второй – специфичность этого правила, которая считается как количество совпавших ограничений, установленных для файла.

Дальше идёт условие, которое проверяет, содержится ли в списке

правило с действием `IGNORE`. Если оно есть – метод возвращает пустой список, так как к этому файлу никакие правила применятся не должны.

```
if self._is_ignore_rule_exist(all_suitable_rules):  
    return []
```

Далее необходимо определить, есть ли среди наиболее специфичных правил, правило с действием удаления файла - `DELETE_FILE`. Для этого полученный список правил с помощью приватного метода `_sort_by_specific_level` сортируется по убыванию специфичности правил, метод `_get_top_specific_rules` возвращает правила с наибольшей одинаковой специфичностью, а `_get_rules_by_action_type` с параметрами в виде списка наиболее специфичных правил и типом `DELETE_FILE` возвращает список правил удаления. Если этот список не пустой, искать цепочки правил не имеет смысла – файл всё равно будет удалён. Поэтому для такой ситуации мы возвращаем в списке одно из правил, которое приведёт к удалению файла:

```
if len(delete_rules) != 0:  
    return [ delete_rules[0] ]
```

Если правила для данного файла не игнорируются и файл не удаляется – следующим шагом будет построение цепочки правил, действия для которых можно было бы выполнить последовательно. Например, может выстроиться цепочка правил, в которой последовательность действий будет такой: переименование файла, перемещение его в подкаталог, и отнесение его к одной из групп в этом подкаталоге. Для того что бы выстроить подобную цепочку правил, используется приватный метод `_get_final_rules_sequence`, в который передаются правила, отсортированные по специфичности. Цепочка строится по следующим правилам:

- 1) Итоговая цепочка правил всегда отсортирована по убыванию специфичности. Это необходимо для того что бы вначале применялись действия, рассчитанные на более узкий круг файлов.

- 2) Список правил оптимизируется по приоритетам. Если для файла существует несколько правил с похожими или перекрывающимися друг друга действиями и разной специфичностью – выбирается правило с большей специфичностью, а все остальные игнорируются.

- 3) Если существует несколько правил с одинаковой специфичностью, применение которых приведёт к одному и тому же результату – в результирующую цепочку попадает одно такое правило, все остальные игнорируются.

- 4) Если правила обладают одинаковой специфичностью, и действия, заданные этими правилами, не конфликтуют между собой – они сортируются

по дате создания.

Цепочка, построенная по данным правилам и представленная в виде списка, и является результатом работы метода `get_rules`.

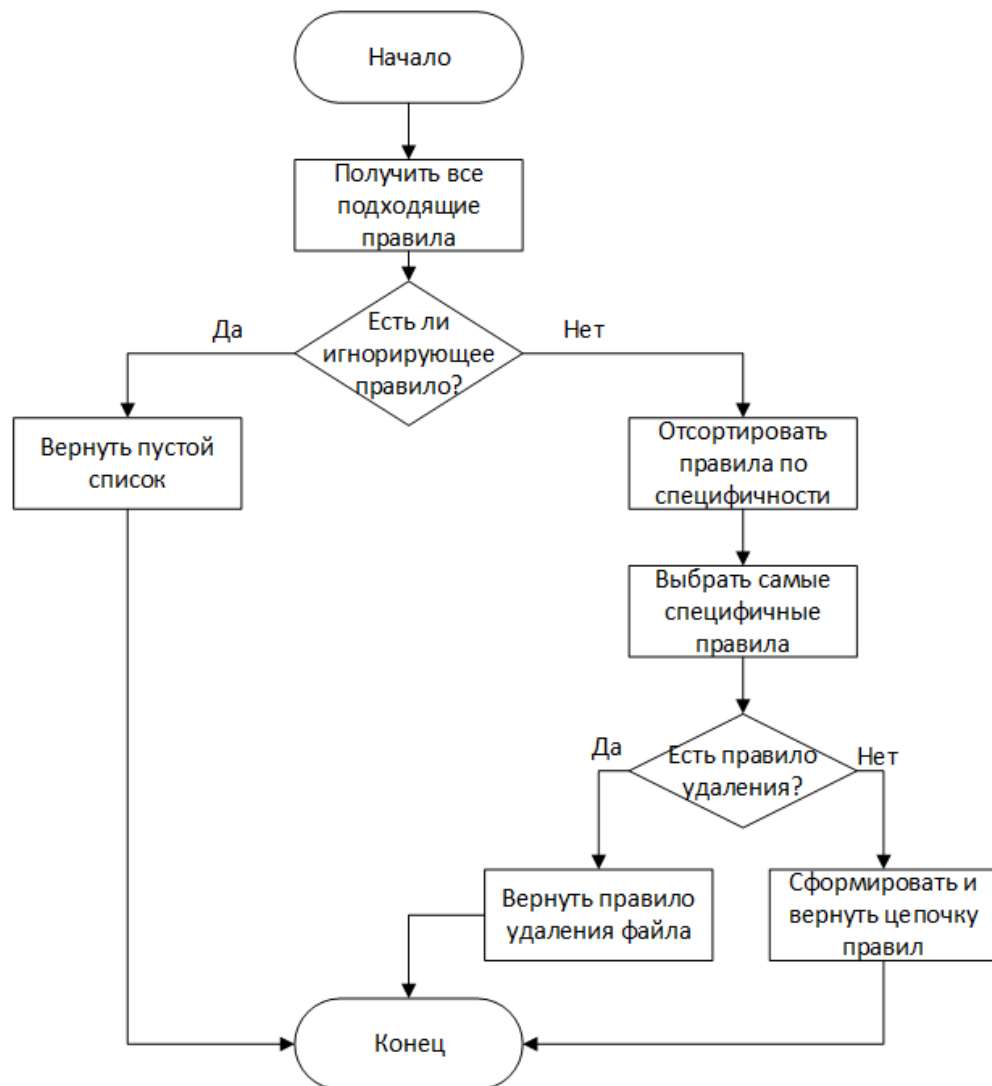


Рисунок 4.4 – Блок-схема алгоритма метода `get_rules`

В данном разделе были подробно описаны алгоритмы основных функций и методов приложения. Алгоритм работы всего приложения можно посмотреть на (ссылка на схему программы).

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – это процесс исследования программного обеспечения, который преследует 2 цели:

- проверить программный продукт на соответствие заявленным требованиям;
- выявить условия, в которых поведение программы является ошибочным либо не соответствует спецификации;

Тестирование программного обеспечения обычно проводится в несколько этапов и является таким же важным этапом разработки программного обеспечения, как и написание кода. Хорошо отлаженный процесс тестирования позволяет как обнаружить уже существующие ошибки, так и не допустить появления новых в уже отлаженном рабочем коде.

Помимо рисков, связанных с неизбежным появлением ошибок в программном коде, существует также риск написать программу, не соответствующую заявленным к ней требованиям. С этой стороны, целью тестирования является выявление несоответствия функционирования программы заявленным требованиям, и устранение этих несоответствий с целью повышения качества.

Тестирование проводилось в 3 этапа:

- модульное тестирование, которое проводилось в процессе написания программного кода;
- автоматическое интеграционное тестирование, которое проводилось после того как все модули программы были в рабочем состоянии;
- полное функциональное тестирование программы после окончания написания программного кода;

Все три этапа важны для обеспечения качества кода. Модульное тестирование позволяет описать желаемое поведение небольших частей кода, функций и методов, и даёт программисту уверенность в том, что ошибок в протестированной части кода нет. Также модульное тестирование позволяет быстро удостовериться в том, что после внесения изменений в код программы он всё ещё работает корректно. Недостаток модульного тестирования в том, что он не гарантирует корректную работу всех модулей вместе. Для того что бы дать программисту такую гарантию, нужны автоматические интеграционные тесты, выполнение которых занимает чуть больше времени и требует работоспособности всех модулей программы. Полное функциональное тестирование программного обеспечения позволяет выявить те недостатки, которые нельзя выявить на предыдущих этапах тестирования, а именно:

- неудобность использования программы для пользователя;
- медленная работа программы в рамках наиболее частых сценариев взаимодействия программы и пользователя;
- побочные эффекты после выполнения программой той или иной

команды пользователя;

Тестирование программы проводилось на следующих машинах:

1) AMD Phenom II X4 965 4 ядра по 3,4 ГГц, оперативная память 8Гб, видеокарта nVidia GTX260 на 1Гб. Диск HDD на 1Тб, скорость - 7200 об/мин. Операционная система - Manjaro Linux.

2) Intel Core i5-4200 4 ядра по 2,4 ГГц, оперативная память 8 Гб, видеокарта nVidia GeForce 960M. Диск SSD на 500 Гб. Операционная система - Ubuntu Linux.

3) Intel Core i7-3770 8 ядер по 3,4 ГГц, оперативная память 16 Гб, видеокарта nVidia GeForce 960M. Диск HDD на 1Тб, скорость – 5200 об/мин. Операционная система – OpenSUSE Linux.

5.1 Модульное тестирование

Модульное тестирование помогает проверить корректность программы путём проверки поведения небольших участков кода, таких как методы и функции. Тест одного из аспектов поведения модуля обычно состоит из трёх этапов:

- моделирование окружения;
- произведение запланированного действия;
- проверка результатов этого действия;

Наиболее сложным этапом создания модульного теста является моделирование окружения. Поскольку каждый тест должен быть изолирован от остальных, окружение настраивается также индивидуально. При настройке задаются все начальные условия, подменяются возвращаемые значения для вызываемых модулем функций, определяются параметры для вызываемого модуля, если они имеют сложную структуру и т.д. Далее над модулем производится действие, например, вызов метода с заранее заданными параметрами. И в завершении проверяется результат вызова – это может быть, как проверка возвращаемого значения метода или функции, так и проверка состояния окружения, если работа метода или функции имеет запланированный побочный эффект.

Написав серию тестов, по одному на каждый аспект работы модуля, можно сказать что модуль покрыт тестами. При внесении в него изменений можно проверить, нарушили ли новые изменения уже отлаженную логику или нет.

Каждый из популярных языков программирования имеет свою библиотеку для проведения модульного тестирования. В языке Python за это отвечает модуль unittest, который и используется для тестирования данного приложения. Пример класса, содержащего тесты для методов класса CommandsDescriptionModule:

```
class CommandsDescriptionModuleTest(
    unittest.TestCase):
```

```

def setUp(self):
    self.commands_module = CommandsDescriptionModule()

def test_executing_none_command(self):
    with self.assertRaises(ValueError):
        self.commands_module.execute_command(None)

def test_not_implemented_command(self):
    with self.assertRaises(NotImplementedError):
        bad_command = Command(tbt.EVENT_LOG_BLOCK, \
                               att.CREATE, None)
        self.commands_module.execute_command(\
            bad_command)

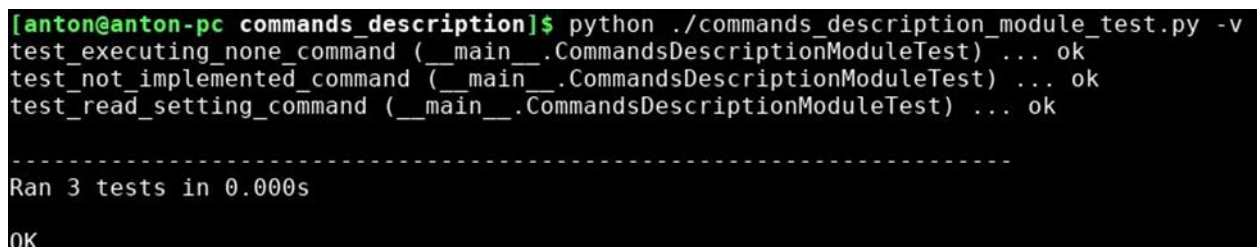
def test_read_setting_command(self):
    read_command = Command(tbt.SETTINGS_BLOCK, \
                           att.READ, None)
    setting = self.commands_module.execute_command(\
        read_command)
    key, value = setting

    self.assertIs(type(setting), tuple)
    self.assertIsNotNone(key)
    self.assertIsNotNone(value)

```

В методе `setUp` задаются общие начальные условия для всех тестов, которые содержатся в данном классе как методы. Этот метод выполняется перед выполнением каждого теста. Методами формата `assert*` проверяются результаты выполнения действий.

Если запустить модуль, содержащий данный класс тестов, в консоль выводится результат выполнения всех тестов в классах этого модуля. В случае класса `CommandsDescriptionModuleTest` вывод представлен на рисунке 5.1. Тесты также можно запускать и по отдельности, но делается это обычно если их настолько много, что их полное выполнение занимает длительное время.



```

[anton@anton-pc commands_description]$ python ./commands_description_module_test.py -v
test_executing_none_command (__main__.CommandsDescriptionModuleTest) ... ok
test_not_implemented_command (__main__.CommandsDescriptionModuleTest) ... ok
test_read_setting_command (__main__.CommandsDescriptionModuleTest) ... ok
-----
Ran 3 tests in 0.000s
OK

```

Рисунок 5.1 – Вывод результата запуска модульных тестов

5.2 Интеграционное тестирование

Интеграционное тестирование – это фаза тестирования, при которой модули объединяются и тестируются в группе. Обычно интеграционное

тестирование следует за модульным и предшествует полному системному тестированию. Интеграционное тестирование позволяет выявить ошибки взаимодействия модулей, задать набор приёмочных тестов, без прохождения которых невозможен выпуск приложения и протестировать приложение под нагрузкой.

Поскольку данное приложение является тремя взаимодействующими между собой программами, цель которых в автоматическом режиме распределять файлы по заданным пользователем правилам – интеграционное тестирование имеет для него первостепенную значимость. Интеграционные тесты для данного приложения написаны на языке оболочки `bash`. Интеграционный тест, так же, как и модульный состоит из задания начальных условий, действия и проверки результата. Примеры интеграционных тестов для проверки правил распределения файлов выглядят следующим образом:

```
# Test 24. Permanently delete file if name changed

mkdir ./test
touch ./test/file.txt
tars add_rule -t /home/anton/Tars/Program/test/ \
    --event-types FILE_NAME_CHANGED \
    --extention-template txt \
    --action-type DELETE_FILE --is-permanent
mv ./test/file.txt ./test/file2.txt

if [ -f ./test/file2.txt ]
then
    echo "ERROR. File ./test/file2.txt was not
    deleted."
else
    echo "SUCCESS. File ./test/file2.txt was deleted."
fi

rm -rf ./test

# Test 25. Peplace file in subfolder if file moved
from

mkdir -p ./test/sub
touch file.txt
tars add_rule -t /home/anton/Tars/Program/test/ \
    --event-types FILE_INCLUDED \
    --extention-template txt \
    --action-type REPLACE_FILE \
    --target-path /home/anton/Tars/Program/test/sub/
mv file.txt ./test/

if ! [ -f ./test/sub/file.txt ]
then
    echo "ERROR. File file.txt was not found in
```

```

        ./test/sub/."
else
    echo "SUCCESS. File file.txt was not found in
        ./test/sub/."
fi

rm -rf ./test/

```

В этих двух тестах вначале создаётся директория, действия в которой будут отслеживаться приложением, создаётся тестовый файл, задаётся правило распределения и над тестовым файлом производится действие таким образом, чтобы он попал под заданное правило. После того как приложение среагировало на событие в файловой системе – в bash-скрипте идёт проверка того что стало с файлом. В зависимости от результатов проверки в консоль выводится либо сообщение об успешно выполненной операции, либо с сообщением об ошибке. Таким образом, имея доступ управлению программой из консоли, можно эмулировать практически любые действия пользователя.

5.3 Полное функциональное тестирование программы

Полное функциональное тестирование приложения проводится после того как код программы полностью написан, все модульные и интеграционные тесты пройдены и нужно решить можно ли продавать\передавать приложение его заказчику или конечному пользователю. Полное тестирование программы проводится людьми, чтобы понять соответствует ли программа заявленным требованиям, какие недостатки имеет интерфейс пользователя, не проявляется ли просадка производительности при типичных сценариях взаимодействия и не оставляет ли работа приложения неучтённых побочных эффектов.

Перед полным тестированием программы составляется спецификация, состоящая из требований, которые оформляются в виде тестовых случаев (Test Cases) и бизнес-процессов, которые описываются в виде типичных сценариев взаимодействия пользователя и программы (Use Cases).

Тестирование проводилось несколькими пользователями на разных компьютерах с разной аппаратной конфигурацией и разными операционными системами. Функциональные тесты, проведённые над программным модулем представлены в таблице 5.1.

Таблица 5.1 – Полное функциональное тестирование программы

Модуль	Содержание теста	Ожидаемый результат	Тест пройден
1	2	3	4
Настройки	Создание настройки без задания значения	Сообщение о некорректной команде	Да

Продолжение таблицы 5.1

1	2	3	4
Настройки	Обновление настройки без задания значения	Сообщение о некорректной команде	Да
Настройки	Удаление обязательной настройки приложения	Сообщение о невозможности удалить настройку	Да
Настройки	Экспорт настроек в существующий системный файл	Сообщение о невозможности экспортировать настройки в данный файл	Да
Настройки	Импорт настроек из пустого файла	Сообщение с перечислением обязательных настроек	Да
Распределение файлов	Перемещение в отслеживаемую директорию большого количества файлов разных типов	Файлы распределяются в течении небольшого промежутка времени	Да
Распределение файлов	Перемещение в отслеживаемую директорию файлов, которые должны игнорироваться	С перемещёнными файлами ничего не происходит	Да
Распределение файлов	Создание правила распределения без обязательных аргументов	Сообщение с перечислением пропущенных аргументов	Да
Распределение файлов	Удаление и создание отслеживаемой директории	Правила для директории удаляются после удаления директории	Да
Распределение файлов	Импорт правил из пустого файла	Сообщение об ошибке импорта с указанием того что файл пуст	Да

Продолжение таблицы 5.1

1	2	3	4
Анализ файлов	Изменения расширения файла на расширение файла с другим типом контента	Расширение не влияет на распознавание типа контента	Да
Анализ файлов	Изменение метаданных файла	Файл распределяется по новым метаданным	Да
Анализ файлов	Изменение размера файла	Файл распределяется по новому размеру	Да
Анализ файлов	Задание правила для удаления системного файла	Файл распознаётся как системный, в файл журнала записывается ошибка о невозможности применить правило к системному файлу	Да
Анализ файлов	Создание директории, которая подпадает под одно из правил только для регулярных файлов	Директория распознаётся, правило к ней не применяется	Да

Как видно из таблицы, приложение прошло все функциональные тесты, что говорит о его готовности к выпуску.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Требования к аппаратному и программному обеспечению

Для работы приложения персональный компьютер пользователя должен обладать следующими характеристиками и набором библиотек:

- операционная система на базе ядра Linux;
- интерпретатор языка программирования Python версии 3.0 и выше;
- 1 Гб оперативной памяти;
- жёсткий диск со скоростью 5200 об\мин и выше или твердотельный накопитель;
- процессор с частотой 2.4 ГГц и выше;

6.2 Руководство по установке системы

Для установки приложения достаточно найти его с помощью пакетного менеджера, установленного на операционной системе пользователя и запустить команду установки, которая будет различаться в зависимости от используемого пакетного менеджера. Часто определённой операционной системе на базе ядра Linux соответствует определённый пакетный менеджер, поэтому далее будет описан процесс установки приложения на наиболее популярных операционных системах.

Для операционных систем на базе Debian, таких как собственно Debian, Ubuntu (и производные от Ubuntu), Linux Mint команда установки приложения будет следующей:

```
# apt-get install tars
```

Для операционных систем, продвигаемых компанией RedHat, таких как RedHat и Fedora:

```
# dnf install tars
```

Для серверной операционной системы CentOS:

```
# yum install tars
```

Для операционных систем на базе Arch-Linux, таких как Arch Linux и Manjaro Linux:

```
# pacman -S tars
```

После выполнения данной команды на компьютер пользователя будет установлено три программы:

- 1) Демон tars-monitor, представляющий монитор файловой системы.
- 2) Демон tars-worker, представляющий распределитель файлов.
- 3) Консольный клиент tars.

6.3 Руководство по использованию программного средства

Пользователь взаимодействует с приложением через консольный клиент. Для того что бы ознакомиться с тем, с чем предстоит работать, у всех программ с консольным интерфейсом по умолчанию есть две команды – help и version. Команда help отображает справочную информацию: описание программы, список доступных команд с описанием их работы и необходимых аргументов для каждой из них, возможные варианты использования команд. Вывод команды help для данного приложения можно видеть на рисунке 6.1.

Знание информации о версии программы может быть необходимо при поиске информации об ошибке, которая случилась во время её работы. Вывод команды version можно видеть на рисунке 6.2.

```

[anton@anton-pc ~]$ tars --help
usage: TARS [-h] [--version]
           {create_setting,read_setting,update_setting,delete_setting,import_settin
s,execute_action,start_monitor,stop_monitor}
           ...

positional arguments:
  {create_setting,read_setting,update_setting,delete_setting,import_settings,export
action,start_monitor,stop_monitor}
    create_setting      command for settings creating
    read_setting        command for settings reading
    update_setting      command for settings updating
    delete_setting     command for settings deleting
    import_settings     command for settings importing
    export_settings     command for settings exporting
    read_log            command for event log reading
    restore             command for file system restoring by event log records
    create_rule         command for rule creating
    read_rule           command for rule reading
    update_rule         command for rule updating
    delete_rule        command for rule deleting
    import_rules        command for rule importing
    export_rules        command for rule exporting
    execute_action      command for immeditely action executing
    start_monitor       command for monitor starting
    stop_monitor        command for monitor stoping

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
[anton@anton-pc ~]$

```

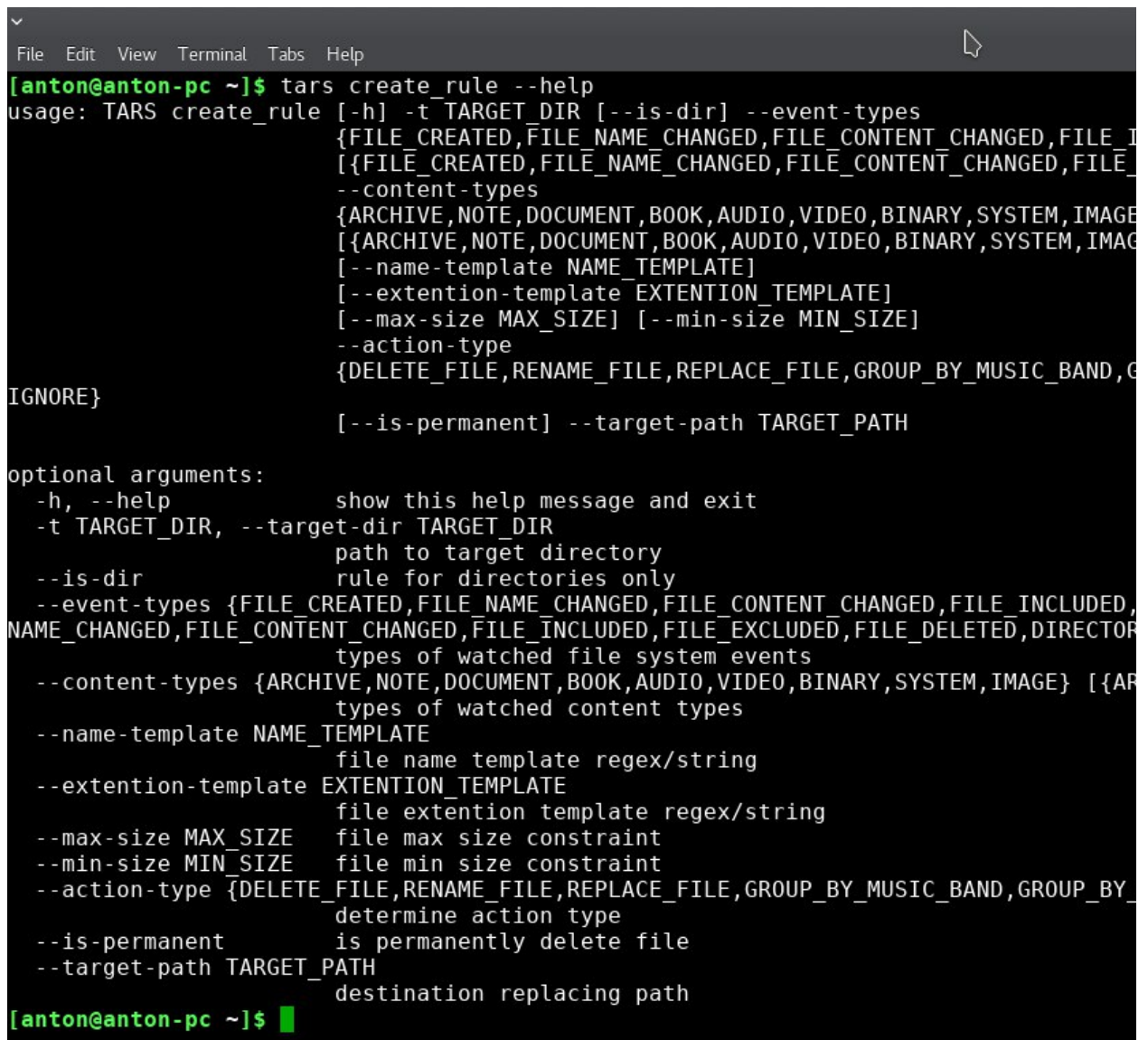
Рисунок 6.1 – Вывод команды help



```
Terminal - anton@anton-pc~  
File Edit View Terminal Tabs Help  
[anton@anton-pc ~]$ tars --version  
TARS 0.5  
[anton@anton-pc ~]$
```

Рисунок 6.2 – Вывод команды version

Для того что бы посмотреть справку по отдельной команде, после имени команды можно также напечатать ключ help. Вывод справки для команды создания правила представлен на рисунке 6.3.

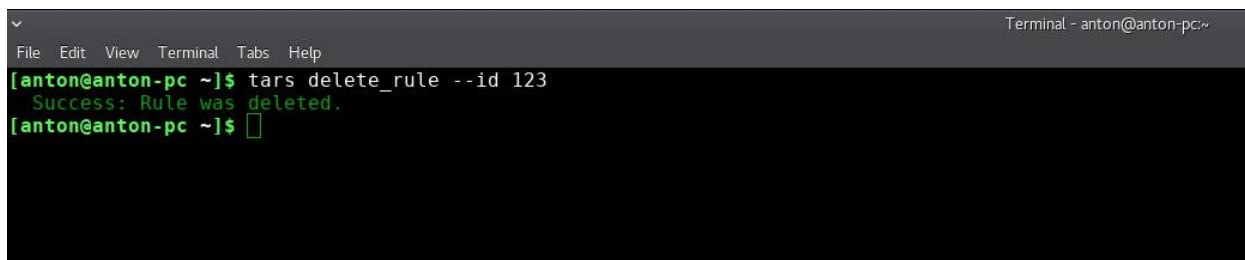


```
File Edit View Terminal Tabs Help  
[anton@anton-pc ~]$ tars create_rule --help  
usage: TARS create_rule [-h] -t TARGET_DIR [--is-dir] --event-types  
                        {FILE_CREATED,FILE_NAME_CHANGED,FILE_CONTENT_CHANGED,FILE_I  
                        [{FILE_CREATED,FILE_NAME_CHANGED,FILE_CONTENT_CHANGED,FILE_  
                        --content-types  
                        {ARCHIVE,NOTE,DOCUMENT,BOOK,AUDIO,VIDEO,BINARY,SYSTEM,IMAGE  
                        [{ARCHIVE,NOTE,DOCUMENT,BOOK,AUDIO,VIDEO,BINARY,SYSTEM,IMAG  
                        [--name-template NAME_TEMPLATE]  
                        [--extention-template EXTENTION_TEMPLATE]  
                        [--max-size MAX_SIZE] [--min-size MIN_SIZE]  
                        --action-type  
                        {DELETE_FILE,RENAME_FILE,REPLACE_FILE,GROUP_BY_MUSIC_BAND,G  
                        IGNORE}  
                        [--is-permanent] --target-path TARGET_PATH  
  
optional arguments:  
  -h, --help            show this help message and exit  
  -t TARGET_DIR, --target-dir TARGET_DIR  
                        path to target directory  
  --is-dir              rule for directories only  
  --event-types {FILE_CREATED,FILE_NAME_CHANGED,FILE_CONTENT_CHANGED,FILE_INCLUDED,  
NAME_CHANGED,FILE_CONTENT_CHANGED,FILE_INCLUDED,FILE_EXCLUDED,FILE_DELETED,DIRECTOR  
                        types of watched file system events  
  --content-types {ARCHIVE,NOTE,DOCUMENT,BOOK,AUDIO,VIDEO,BINARY,SYSTEM,IMAGE} [{AR  
                        types of watched content types  
  --name-template NAME_TEMPLATE  
                        file name template regex/string  
  --extention-template EXTENTION_TEMPLATE  
                        file extention template regex/string  
  --max-size MAX_SIZE  file max size constraint  
  --min-size MIN_SIZE  file min size constraint  
  --action-type {DELETE_FILE,RENAME_FILE,REPLACE_FILE,GROUP_BY_MUSIC_BAND,GROUP_BY_  
                        determine action type  
  --is-permanent       is permanently delete file  
  --target-path TARGET_PATH  
                        destination replacing path  
[anton@anton-pc ~]$
```

Рисунок 6.3 – Вывод справки для команды создания правила

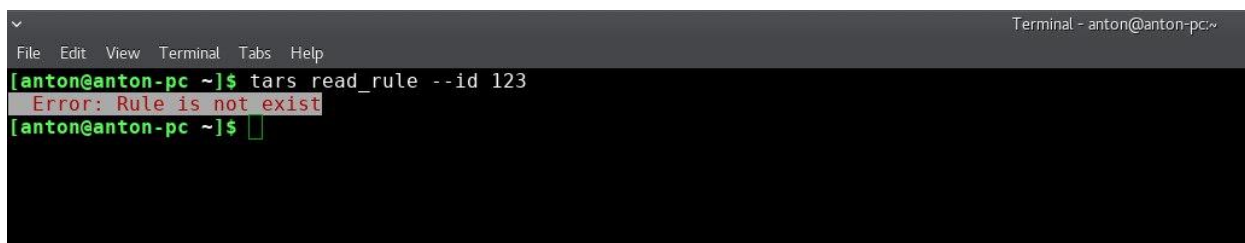
Примеры успешного выполнения команды и выполнения команды с ошибкой можно видеть на рисунке 6.4 и рисунке 6.5 соответственно. В разных статусах выполнения команд результаты выполнения выводятся в консоль

разным цветом.

A terminal window titled "Terminal - anton@anton-pc:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is [anton@anton-pc ~]\$. The command 'tars delete_rule --id 123' is entered. The output is 'Success: Rule was deleted.' in green text. The prompt returns to [anton@anton-pc ~]\$.

```
[anton@anton-pc ~]$ tars delete_rule --id 123
Success: Rule was deleted.
[anton@anton-pc ~]$
```

Рисунок 6.4 – Пример успешного выполнения команды

A terminal window titled "Terminal - anton@anton-pc:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is [anton@anton-pc ~]\$. The command 'tars read_rule --id 123' is entered. The output is 'Error: Rule is not exist' in red text. The prompt returns to [anton@anton-pc ~]\$.

```
[anton@anton-pc ~]$ tars read_rule --id 123
Error: Rule is not exist
[anton@anton-pc ~]$
```

Рисунок 6.5 – Пример выполнения команды с ошибкой

Зная информацию, проведённую выше, пользователь может управлять работой приложения изучая справки к отдельным командам и применяя их на практике.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ФАЙЛОВОГО МЕНЕДЖЕРА С ФУНКЦИЕЙ АВТОМАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ФАЙЛОВ

7.1 Характеристика программного продукта

Программный продукт предназначен для автоматического распределения файлов на персональном компьютере пользователя. Этот продукт позволяет задать правила для распределения различных типов файлов в различных директориях, в том числе и специфические правила, рассчитанные на музыкальные файлы, видео и книги. Всю работу по отслеживанию и применению этих правил программа берёт на себя. В данном приложении очень большая роль уделена производительности, в частности пользователь может самостоятельно задать настройки, влияющие на производительность приложения на целевом компьютере. Имеется возможность быстрого создания интерфейса пользователя под нужды пользователей конкретной операционной системы, чем обеспечивается гибкость разработки и экономическая ценность продукта. В настоящее время аналоги данного продукта представляют собой узкоспециализированные решения, созданные программистами для автоматизации своих повседневных задач. Особенностью данной разработки является широкий охват пользователей за счёт переносимости между операционными системами, что позволяет значительно увеличить прибыль за продажу продукта.

Разработка данного продукта осуществляется в IT-компании «Техартгрупп» для продажи копиями. Исходя из маркетингового исследования, лицензии на программный продукт будут востребованы на рынке в течении 4 лет; в период с 2017 г. По 2020 г. Планируется продавать по 1000 лицензий в год.

Разработка и внедрение данного программного продукта позволяет:

- разработать интерфейс продукта под конкретную операционную систему;
- снять с пользователя рутинную работу по поддержанию чистоты в рабочих директориях;
- задавать правила для автоматического распределения различных типов файлов;
- обеспечивать простоту поддержки и дальнейшего развития;

Экономическая целесообразность инвестиций в разработку и реализацию представленного программного продукта определяется на основе расчёта таких показателей, как:

- чистый дисконтированный доход продукта;
- срок окупаемости инвестиций программного продукта;
- рентабельность инвестиций в разработку программного продукта;

7.2 Расчёт сметы затрат и отпускной цены программного продукта

Основная заработная плата (Z_o) исполнителей определяется по формуле:

$$Z_o = \sum_{i=1}^n T_{\text{чи}} \cdot T_{\text{ч}} \cdot \Phi_{\text{эи}} \cdot K, \quad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПС;

$T_{\text{чи}}$ – часовая тарифная ставка i -го исполнителя (руб.);

$\Phi_{\text{эи}}$ – эффективный фонд рабочего времени i -го исполнителя (дней);

$T_{\text{ч}}$ – количество часов работы в день (ч);

K – коэффициент премирования.

Примем тарифную ставку 1-го разряда равной 190,00 рублей. Среднемесячная норма рабочего времени составляет 168 часов. Часовой тарифный оклад руководителя проекта составляет $190 \cdot 3,25/168 = 3,68$ рубля. Часовой тарифный оклад инженера-программиста составляет $190 \cdot 2,48/168 = 2,80$ рубля. Часовой тарифный оклад инженера-программиста составляет $190 \cdot 2,27/168 = 1,97$ рубля. Премия сотрудникам начисляется компанией в размере 25% от итоговой суммы.

Результаты расчёта основной заработной платы исполнителей представлены в таблице 7.1.

Таблица 7.1 - Расчёт основной заработной платы

Исполнитель	Раз- ряд	Тариф- ный коэффи- циент	Месяч- ная тарифная ставка, руб.	Часовая тарифная ставка, руб.	Плано- вый фонд рабочего времени, дн.	Заработ- ная плата, руб.
Руководи- тель проекта	12	3,25	617,50	3,68	30	667,12
Инженер- программист	10	2,48	471,20	2,80	90	1520,49
Инженер- программист	10	2,48	471,20	2,80	90	1520,49
Тестирующий	5	1,74	330,60	1,97	60	711,73
Итого						4419,84
Премия (25%)						1104,96
Основная заработная плата (Z_o)						5524,80

Дополнительная заработная плата ($З_д$) рассчитывается по формуле:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (7.2)$$

где $Н_д$ – норматив дополнительной заработной платы, 15%.

Размер дополнительной заработной платы исполнителей составит:

$$З_д = \frac{5524,80 \cdot 15}{100} = 828,72 \text{ руб.}$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ($З_{оз}$) определяется в соответствии с действующими законодательными актами по формуле:

$$P_{\text{соц}} = \frac{З_о + З_д}{100} \cdot Н_{\text{соц}}, \quad (7.3)$$

где $Н_{\text{соц}}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, 34 + 0,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составит:

$$P_{\text{соц}} = \frac{5524,80 + 828,72}{100} \cdot 34,6 = 2198,31 \text{ руб.}$$

Расходы по статье «Машинное время» ($P_{\text{мв}}$) включают оплату машинного времени, необходимого для разработки и отладки ПС и определяется по формуле:

$$P_{\text{мв}} = Ц_м \cdot T_ч \cdot C_p, \quad (7.4)$$

где $Ц_м$ – цена одного машино-часа, руб.;

$T_ч$ – количество часов работы в день, ч.;

C_p – длительность проекта, дн.

Стоимость одного машино-часа на предприятии составляет 1,50 рублей. Разработка проекта займёт 90 дней. Определим затраты по статье «Машинное время»:

$$P_{\text{мв}} = 1,50 \cdot 8 \cdot (90 + 90 + 60) = 2880,00 \text{ руб.}$$

Расходы по статье «Прочие затраты» ($P_{\text{пз}}$) включают затраты на

приобретение специальной научно-технической информации и специальной литературы. Определяется в процентах к основной заработной плате:

$$P_{пз} = \frac{З_o \cdot H_{пз}}{100}, \quad (7.5)$$

где $H_{пз}$ – норматив прочих затрат в целом по организации, %.

$$P_{пз} = \frac{5524,80 \cdot 50}{100} = 2762,40 \text{ руб.}$$

Общая сумма расходов по всем статьям на ПО ($C_{п}$) представляет полную себестоимость ПО:

$$C_{п} = P_{мв} + З_o + З_d + P_{пз}, \quad (7.6)$$

$$C_{п} = 2880,00 + 5524,80 + 828,72 + 828,72 + 2498,31 + 2764,40 = 14196,23 \text{ руб.}$$

Прогнозируемая прибыль рассчитывается по формуле:

$$\Pi_o = \frac{C_{п} \cdot Y_p}{100}, \quad (7.7)$$

где Y_p – уровень рентабельности, 50%.

$$\Pi_o = \frac{14196,23 \cdot 50}{100} = 7098,11 \text{ руб.}$$

Прогнозируемая цена без налогов (цена предприятия $\Pi_{п}$) рассчитывается по формуле:

$$\Pi_{п} = C_{п} + \Pi_o, \quad (7.8)$$

$$\Pi_{п} = 7098,11 + 14196,23 = 21294,34 \text{ руб.}$$

Налог на добавленную стоимость (НДС) рассчитывается по формуле:

$$\text{НДС} = \frac{\Pi_{п} \cdot H_{дс}}{100}, \quad (7.9)$$

где $H_{дс}$ – ставка налога на добавленную стоимость, 20%.

$$\text{НДС} = \frac{21294,34 \cdot 20}{100} = 4258,87 \text{ руб.}$$

Прогнозируемая отпускная цена ($\Pi_{\text{от}}$) рассчитывается по формуле:

$$\Pi_{\text{от}} = C_{\text{п}} + \Pi_{\text{о}} + \text{НДС}, \quad (7.10)$$

$$\Pi_{\text{от}} = 7098,11 + 14196,23 + 4258,87 = 25553,21 \text{ руб.}$$

При расчёте чистой прибыли ($\Pi_{\text{ч}}$) для резидентов ПВТ налог на прибыль не учитывается:

$$\begin{aligned} \Pi_{\text{ч}} &= \Pi_{\text{о}}, \\ \Pi_{\text{ч}} &= 7098,11 \text{ руб.} \end{aligned} \quad (7.11)$$

Расчёты сметы затрат и отпускной цены можно свести в таблицу 7.2.

Таблица 7.2 – Смета затрат и отпускная цена программного продукта

Наименование статей	Условные обозначения	Значение, руб.	Методика расчёта
1	2	3	4
Основная заработная плата исполнителей	$З_{\text{о}}$	5524,80	$З_{\text{о}} = \sum_{i=1}^n T_{\text{ч}i} \cdot T_{\text{ч}} \cdot \Phi_{\text{э}i} \cdot K$
Дополнительная заработная плата исполнителей	$З_{\text{д}}$	828,72	$З_{\text{д}} = \frac{З_{\text{о}} \cdot H_{\text{д}}}{100}$
Отчисления в фонд социальной защиты населения	$P_{\text{соц}}$	34,60	$P_{\text{соц}} = \frac{З_{\text{о}} + З_{\text{д}}}{100} \cdot H_{\text{соц}}$
Машинное время	$P_{\text{мв}}$	2880,00	$P_{\text{мв}} = \Pi_{\text{м}} \cdot T_{\text{ч}} \cdot H_{\text{соц}}$
Прочие прямые расходы	$P_{\text{пз}}$	2462,40	$P_{\text{пз}} = \frac{З_{\text{о}} \cdot H_{\text{пз}}}{100}$
Полная себестоимость	$C_{\text{п}}$	14196,23	$C_{\text{п}} = P_{\text{мв}} + З_{\text{о}} + З_{\text{д}} + P_{\text{пз}}$
Прогнозируемая прибыль	$\Pi_{\text{о}}$	7098,11	$\Pi_{\text{о}} = \frac{C_{\text{п}} \cdot Y_{\text{п}}}{100}$
Прогнозируемая цена без налогов	$\Pi_{\text{п}}$	21294,34	$\Pi_{\text{п}} = C_{\text{п}} + \Pi_{\text{о}}$
НДС	НДС	4258,87	$\text{НДС} = \frac{\Pi_{\text{п}} \cdot H_{\text{дс}}}{100}$

Продолжение таблицы 7.1

1	2	3	4
Прогнозируемая отпускная цена	$\Pi_{от}$	25553,21	$\Pi_{от} = C_{п} + \Pi_{о} + НДС$

7.3 Расчёт экономического эффекта ПО от свободной реализации на рынке

Расчёт цены на одну копию (лицензию) ПО. Цена формируется на основе затрат на разработку и реализацию ПО (затраты на реализацию можно принять в пределах 5-10% от затрат на разработку) и запланированного уровня рентабельности (Y_p):

$$\Pi = \frac{Z_p}{N} + \Pi_{ед} + НДС, \quad (7.12)$$

где Π – цена реализации одной копии (лицензии) ПО, руб.

Z_p – сумма расходов на разработку и реализацию, руб.

N – количество копий (лицензий) ПО, которое будет куплено клиентами за год;

$\Pi_{ед}$ – прибыль, получаемая организацией-разработчиком от реализации одной копии программного продукта, руб.;

НДС – сумма налога на добавленную стоимость, руб.

$$\Pi_{ед} = \frac{C_{п} \cdot Y_p}{N \cdot 100}, \quad (7.13)$$

где Y_p – запланированный норматив рентабельности.

$$\Pi_{ед} = \frac{14196,23 \cdot 50}{1000 \cdot 100} = 7,09 \text{ руб.}$$

Расчёт цены на одну копию (лицензию) ПО:

$$\Pi = \frac{C_{п} + НДС}{N} + \Pi_{ед}, \quad (7.14)$$

$$\Pi = \frac{14196,23 + 4258,87}{1000} + 7,09 = 25,54 \text{ руб.}$$

Общие капитальные вложения (K_o) потребителя, связанные с приобретением, внедрением и использованием ПО рассчитывается по

формуле:

$$K_o = K_{\text{пр}} + K_c + K_{\text{ос}} + K_{\text{об}}, \quad (7.15)$$

где $K_{\text{пр}}$ – затраты пользователя на приобретение по цене на одну копию, руб.;

K_c – затраты пользователя на оплату услуг по сопровождению ПО (10% от $K_{\text{пр}}$), руб.;

$K_{\text{ос}}$ – затраты пользователя на освоение ПО (10% от $K_{\text{пр}}$), руб.;

$K_{\text{об}}$ – затраты на пополнение оборотных средств (15% от $K_{\text{пр}}$), руб.

$$K_o = 25,54 \cdot (1,00 + 0,10 + 0,10 + 0,15) = 34,48 \text{ руб.}$$

А суммарная годовая прибыль (Π) по проекту ежегодно будет равна:

$$\Pi = \Pi_{\text{ед}} \cdot N, \quad (7.16)$$

$$\Pi = 7,09 \cdot 1000 = 7090,00 \text{ руб.}$$

Так как компания «Техартгрупп» является резидентом ПВТ и налог на прибыль не взимается, то прибыль, которая остаётся в распоряжении предприятия ($\Delta\Pi_{\text{ч}}$), равна суммарной годовой прибыли:

$$\Delta\Pi_{\text{ч}} = \Pi, \quad (7.17)$$

$$\Delta\Pi_{\text{ч}} = 7090,00 \text{ руб.}$$

7.4 Расчёт показателей экономической эффективности разработки и реализации программного продукта

Полученные суммы результата (чистой прибыли) и затрат (капитальных вложений) по годам необходимо привести к единому моменту времени – расчётному году (2017) путём умножения результатов и затрат на коэффициент дисконтирования α_t , который рассчитывается по формуле:

$$\alpha_t = \frac{1}{(1 + E_n)^{t_i - t_p}}, \quad (7.18)$$

где E_n – норма дисконтирования в долях единицы (0,14)

t_i – порядковый номер года, результаты и затраты которого приводятся к расчётному году;

t_p – расчётный год ($t_p = 1$).

$$\alpha_{t1} = \frac{1}{(1+0,14)^{1-1}} = 1.$$

С учётом того, что проект будет запущен только во второй половине года, то $t_i = t_i/2 = 0,5$.

$$\alpha_{t2} = \frac{1}{(1+0,14)^{2-1}} = 0,88.$$

$$\alpha_{t3} = \frac{1}{(1+0,14)^{3-1}} = 0,769.$$

$$\alpha_{t4} = \frac{1}{(1+0,14)^{4-1}} = 0,675.$$

Показатели экономической эффективности представлены в таблице 7.3.

Таблица 7.3 – Показатели экономической эффективности разработки и реализации программного продукта

Показатели	Условные обознач.	Ед. изм.	Годы			
			2017	2018	2019	2020
Результаты						
Прирост чистой прибыли	$\Delta\Pi_{\text{ч}}$	руб.	3545,00	7090,00	7090,00	7090,00
С учётом фактора времени	$\Delta\Pi_{\text{ч}} * \alpha_t$	руб.	3545,00	6239,20	5452,21	4785,75
Затраты						
Затраты на разработку	$C_{\text{п}}$	руб.	14196,23	-	-	-
Затраты на разработку и реализацию	$З_{\text{р}}$	руб.	14196,23	-	-	-
Экономический эффект						
Превышение результата над затратами	$P_t - З_t$	руб.	-10651,23	-3561,23	3528,77	10618,77
То же с нарастающим итогом	$P_t - З_t \alpha_t$	руб.	-10651,23	-4412,03	1040,18	5825.93
Коэффициент	α_t		1	0,88	0,769	0,675

Рассчитаем рентабельность инвестиций в разработку и внедрение программного продукта ($P_{и}$) по формуле:

$$P_{и} = \frac{\Pi_{\text{чср}}}{Z_p} \cdot 100\%, \quad (7.19)$$

где $\Pi_{\text{чср}}$ – среднегодовая величина чистой прибыли за расчетный период, руб., которая определяется по формуле:

$$\Pi_{\text{чср}} = \frac{\sum_{i=1}^n \Pi_{\text{чи}}}{n}, \quad (7.20)$$

где $\Pi_{\text{чи}}$ – чистая прибыль, полученная в году t , руб.

$$\Pi_{\text{чср}} = \frac{3545,00 + 6239,20 + 5452,21 + 4785,75}{4} = 5005,54 \text{ руб.}$$

$$P_{и} = \frac{5005,54}{14196,23} \cdot 100 = 35,25\%$$

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

- чистый дисконтированный доход за четыре года работы программы составит 5005,54руб.;
- затраты на разработку программного продукта окупятся на четвертый год его использования;
- рентабельность инвестиций составит 35,25%;

Таким образом, разработка и реализация программного продукта является эффективным и инвестирование средств в разработку продукта целесообразно.

ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта было разработано программное приложение, которое позволяет в автоматическом режиме поддерживать структуру каталогов пользователя в необходимом для него порядке. Данное приложение позволяет пользователю задать правила распределения файлов, попадающих в интересующие его директории и настройки, влияющие на работу приложения на конкретном компьютере.

Много внимания было уделено модульности и переносимости. Приложение разрабатывалось как распределённое, поэтому для того что бы перенести его на другую операционную систему нужно заново реализовывать только некоторые платформозависимые модули, в частности монитор файловой системы и клиент. В ходе разработки были описаны все протоколы и форматы данных для обмена, которые нужно использовать при их реализации.

Также были учтены пользователи, которые имеют несколько персональных компьютеров. Приложение имеет возможность экспорта и импорта правил распределения и настроек, что позволяет пользователю задать их только для одной машины, и затем распространить их на все остальные.

Использование данного приложения позволяет добиться существенного сокращения временных затрат на рутинную работу по самостоятельному перемещению удалению или переименованию файлов. Модульное же устройство приложения позволяет значительно облегчить дальнейшую поддержку существующего функционала и добавление новых возможностей, что позволит привлечь большее количество пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Google Trends [Электронный ресурс]. – Режим доступа : <https://trends.google.ru/trends/explore?q=скачать>
- [2] Яндекс. Подбор слов [Электронный ресурс]. – Режим доступа : <https://wordstat.yandex.by/#!/?words=скачать>
- [3] Google Trends [Электронный ресурс]. – Режим доступа : <https://trends.google.ru/trends/explore?q=download>
- [4] Habrahabr [Электронный ресурс]. – Режим доступа : <https://habrahabr.ru/post/77760/>
- [5] µTorrent. Help page [Электронный ресурс]. – Режим доступа : <http://help.utorrent.com/customer/en/portal/articles/178825-downloading-with-bittorrent>
- [6] Notepad++. Features page [Электронный ресурс]. – Режим доступа : <https://notepad-plus-plus.org/features/>
- [7] DropIt project [Электронный ресурс]. – Режим доступа : <http://www.dropitproject.com/>
- [8] CCleaner [Электронный ресурс]. – Режим доступа : <https://www.piriform.com/ccleaner>
- [9] PCPro100 [Электронный ресурс]. – Режим доступа : <http://pcpro100.info/programmyi-poiska-dublikatov-faylov/>
- [10] Mac App Store [Электронный ресурс]. – Режим доступа : <https://itunes.apple.com/us/app/file2folder/id654460361?mt=12>
- [11] MacOSworld [Электронный ресурс]. – Режим доступа : <http://macosworld.ru/automator-avtomaticheskaya-sortirovka-papki-z/>
- [12] Markus Jais Blog [Электронный ресурс]. – Режим доступа : <http://markusjais.com/linux-file-system-events-with-c-python-and-ruby/>
- [13] MSDN [Электронный ресурс]. – Режим доступа : [https://msdn.microsoft.com/en-us/library/windows/desktop/aa365261\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa365261(v=vs.85).aspx)
- [14] Git [Электронный ресурс]. – Режим доступа : <https://git-scm.com/>
- [15] Керниган, Б. Язык программирования Си : справ. пособие / – Б. Керниган, Д. Ритчи. – М. : Вильямс, 2015. – 304 с.
- [16] IBM developerWorks [Электронный ресурс]. – Режим доступа : https://www.ibm.com/developerworks/ru/library/l-python_part_1/

ПРИЛОЖЕНИЕ А

(обязательное)

Файловый менеджер с функцией автоматического распределения файлов.
Ведомость документов.

ПРИЛОЖЕНИЕ Б

(обязательное)

Файловый менеджер с функцией автоматического распределения файлов.
Спецификация.

ПРИЛОЖЕНИЕ В

(обязательное)

Файловый менеджер с функцией автоматического распределения файлов.

Исходный код.