Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

УТВЕРЖДАЮ
заведующий каф. КИБЭВС
_____А.А. Шелупанов
«_____»_____2014г.

КОМПЬЮТЕРНАЯ ЭКСПЕРТИЗА

Отчет по групповому проектному обучению
Группа КИБЭВС-1208

Ответственный исполнитель
студент гр. 720-1
_____Д.С. Никифоров
«_____»_____2014г.

Научный руководитель
аспирант каф. КИБЭВС
_____А.И. Гуляев
«_____»_____2014г.

2014

РЕФЕРАТ

Курсовая работа содержит 56 страниц, 23 рисунков, 4 таблицы, 14 источников, 1 приложение.

КОМПЬЮТЕРНАЯ ЭКСПЕРТИЗА, ФОРЕНЗИКА, ЛОГИ, QT, XML, GIT, LATEX, ЖУРНАЛЬНЫЕ ФАЙЛЫ, SKYPE, PIDGIN, WINDOWS, СЕРТИФИКАЦИЯ, РЕПОЗИТОРИЙ, XSLT, LIB, БИБЛИОТЕКИ.

Цель работы — создание программного комплекса, предназначенного для проведения компьютерной экспертизы.

Задачей, поставленной на данный семестр, стало написание программного комплекса, имеющего следующие возможности:

1) сбор и анализ событий системных журналов операционной системы;
2) сбор и анализ информации из журналов истории браузеров;
3) сбор и анализ истории переписки мессенджеров;
4) сбор и анализ событий журнальных файлов приложений;
5) поиск файлов по имени.

Результатами работы в данном семестре являются:

– разработка архитектуры проекта;
– использование в разработке системы контроля версий GIT;
– использование Qt — кроссплатформенной библиотеки C++;
– изучение вопроса сертификации для возможности внедрения данного комплекса в гос. структуры, занимающиеся информационной безопасностью;
– использование системы компьютерной вёрстки TEXдля написания документации.

Пояснительная записка выполнена при помощи системы компьютерной вёрстки LATEX.

Список исполнителей

Моргуненко А.В. – документатор.

Никифоров Д.С. – программист, ответственный исполнитель, ответственный за написание части системы, работающей с логами системы.

Поляков И.Ю. – программист, ответственный за написание части системы, работающей с логами мессенджеров.

Пономарёв А.К. – аналитик.

Содержание

Введение

Компьютерно-техническая экспертиза – это самостоятельный род судебных экспертиз, относящийся к классу инженерно-технических экспертиз, проводимых в следующих целях: определения статуса объекта как компьютерного средства, выявление и изучение его роли в рассматриваемом деле, а так же получения доступа к информации на электронных носителях с последующим всесторонним её исследованием [1]. Компьютерная экспертиза помогает получить доказательственную информацию и установить факты, имеющие значение для уголовных, гражданских и административных дел, сопряжённых с использованием компьютерных технологий. Для проведения компьютерных экспертиз необходима высокая квалификация экспертов, так как при изучении представленных носителей информации, попытке к ним доступа и сбора информации возможно внесение в информационную среду изменений или полная утрата важных данных.

Компьютерная экспертиза, в отличие от компьютерно-технической экспертизы, затрагивает только информационную составляющую, в то время как аппаратная часть и её связь с программной средой не рассматривается.

На протяжении предыдущих семестров нами были рассмотрены такие направления компьютерной экспертизы, как исследование файловых систем, сетевых протоколов, организация работы серверных систем, механизм журналирования событий. Также нами были изучены основные задачи, которые ставятся перед сотрудниками правоохранительных органов, которые проводят компьютерную экспертизу, и набор чуществующих утилит, способных помочь эксперту в проведении компьютерной экспертизы. Было выявлено, что существует множество разрозненных программ, предназначенных для просмотра лог-файлов системы и таких приложений, как мессенджеры и браузеры, но для каждого вида лог-файлов необходимо искать отдельную программу. Так как ни одна из них не позволяет эксперту собрать воедино и просмотреть все логи системы, браузеров и мессенджеров, было решено создать для этой цели собственный автоматизированный комплекс, которому на данный момент нет аналогов.

1 Назначение и область применения

Разрабатываемый комплекс предназначен для автоматизации процесса сбора информации с исследуемого образа жёсткого диска.

2 Постановка задачи

На данный семестр были поставлены следующие задачи:

– определить средства для разработки;
– разработать архитектуру проекта;
– реализовать структуру проекта;
– определить единый формат выходных данных;
– реализовать несколько модулей.

Задачи по проектированию модулей:

1) сбор и анализ истории переписки пользователей системы Windows из Pidgin и Skype;
2) написание автоматизированных модулей для сбора истории переписки пользователей си-

стемы Windows из таких программ как Pidgin и Skype;

3) сбор и анализ событий журнальных файлов Windows;

4) написание автоматизированного модуля для сбора событий журнальных файлов Windows.

3 Инструменты

3.1 Система контроля версий Git

Для разработки программного комплекса для проведения компьютерной экспертизы решено использовать Git.

Git — распределённая система управления версиями файлов. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, как противоположность системе управления версиями Subversion (также известная как «SVN») [2].

Необходимость использования системы версий, очевидна. Так как в группе несколько программистов и тестер, мы имеем:

– возможность удаленной работы с исходными кодами;

– возможность создавать свои ветки, не мешая при этом другим разработчикам;

– доступ к последним изменениям в коде, т.к. все исходники хранятся на сервере git.keva.su ;

– исходные коды защищены, доступ к ним можно получить лишь имея RSA-ключ;

– возможность откатиться к любой стабильной стадии проекта.

Основные постулаты работы с кодом в системе Git:

– каждая задача решается в своей ветке;

– коммитим сразу, как что-то получили осмысленное;

– в master мержится не разработчиком, а вторым человеком, который производит вычитку и тестирование изменения;

– все коммиты должны быть осмысленно подписаны/прокомментированы.

Для работы над проектом нами был поднят собственный репозиторий на сервере git.keva.su. Адреса репозиториев следующие:

Исходные файлы проекта:

git clone git@git.keva.su:gpo.git gpo.git

Репозиторий для тестирования проекта:

git clone git@git.keva.su:gpo-testdata.git gpo-testdata.git

3.2 Система компьютерной вёрстки TEX

TEX — это созданная американским математиком и программистом Дональдом Кнутом система для вёрстки текстов. Сам по себе TEX представляет собой специализированный язык программирования.Каждая издательская система представляет собой пакет макроопределений этого языка.

LATEX — это созданная Лэсли Лэмпортом издательская система на базе TEX'а[3]. LATEX позволяет пользователю сконцентрировать свои услия на содержании и структуре текста, не заботясь о деталях его оформления.

Для подготовки отчётной и иной документации нами был выбран LATEX так как совместно с системой контроля версий Git он предоставляет возможность совместного создания и редактирова-

ния документов. Огромным достоинством системы LaTeX то, что создаваемые с её помощью файлы обладают высокой степенью переносимости [4].

Совместно с LaTeX часто используется BibTeX — программное обеспечение для создания форматированных списков библиографии. Оно входит в состав дистрибутива LaTeX и позволяет создавать удобную, универсальную и долговечную библиографию. BibTeX стал одной из причин, по которой нами был выбран LaTeX для создания документации.

3.3 Qt - кроссплатформенный инструментарий разработки ПО

Qt - это кроссплатформенная библиотека C++ классов для создания графических пользовательских интерфейсов (GUI) от фирмы Digia. Эта библиотека полностью объектно-ориентированная, что обеспечивает легкое расширение возможностей и создание новых компонентов. Ко всему прочему, она поддерживает огромнейшее количество платформ.

Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Список использованных классов фраемворка QT

– iostream
– QChar
– QCryptographicHash
– QDir
– QDirIterator
– QFile
– QFileInfo
– QIODevice
– QList
– QRegExp
– QString
– QTextStream
– QtSql/QSqlDatabase
– QVector
– QXmlStreamReader
– QXmlStreamWriter
– Conversations

Класс QXmlStreamWriter представляет собой XML писателя с простым потоковым.

Класс QXmlStreamReader представляет собой быстрый синтаксически корректный XML анализатор с простым потоковым API.

QVector представляет собой класс для создания динамических массивов.

Модуль QtSql/QSqlDatabase помогает обеспечить однородную интеграцию БД в ваши Qt при-

ложения.

Класс QTextStream предоставляет удобный интерфейс для чтения и записи текста.

QTextStream может взаимодействовать с QIODevice, QByteArray или QString. Используя потоковые операторы QTextStream, вы можете легко читать и записывать слова, строки и числа. При формировании текста QTextStream поддерживает параметры форматирования для заполнения и выравнивания полей и форматирования чисел. [5]

Класс QString предоставляет строку символов Unicode.

QString хранит строку 16-битных QChar, где каждому QChar соответствует один символ Unicode 4.0. (Символы Unicode со значениями кодов больше 65535 хранятся с использованием суррогатных пар, т.е. двух последовательных QChar.)

Unicode - это международный стандарт, который поддерживает большинство использующихся сегодня систем письменности. Это расширение US-ASCII (ANSI X3.4-1986) и Latin-1 (ISO 8859-1), где все символы US-ASCII/Latin-1 доступны на позициях с тем же кодом.

Внутри QString использует неявное совместное использование данных (копирование-при-записи), чтобы уменьшить использование памяти и избежать ненужного копирования данных. Это также позволяет снизить накладные расходы, свойственные хранению 16-битных символов вместо 8-битных.

В дополнение к QString Qt также предоставляет класс QByteArray для хранения сырых байт и традиционных нультерминальных строк. В большинстве случаев QString - необходимый для использования класс. Он используется во всем API Qt, а поддержка Unicode гарантирует, что ваши приложения можно будет легко перевести на другой язык, если в какой-то момент вы захотите увеличить их рынок распространения. Два основных случая, когда уместно использование QByteArray: когда вам необходимо хранить сырые двоичные данные и когда критично использование памяти (например, в Qt для встраиваемых Linux-систем).[6]

Класс QRegExp предоставляет сопоставление с образцом при помощи регулярных выражений.

Регулярное выражение, или ”regexp”, представляет собой образец для поиска соответствующей подстроки в тексте. Это полезно во многих ситуациях, например:

Проверка правильности – регулярное выражение может проверить, соответствует ли подстрока каким-либо критериям, например, целое ли она число или не содержит ли пробелов. Поиск – регулярное выражение предоставляет более мощные шаблоны, чем простое соответствие строки, например, соответствие одному из слов mail, letter или correspondence, но не словам email, mailman, mailer, letterbox и т.д. Поиск и замена – регулярное выражение может заменить все вхождения подстроки другой подстрокой, например, заменить все вхождения & на &, исключая случаи, когда за & уже следует amp;. Разделение строки – регулярное выражение может быть использовано для определения того, где строка должна быть разделена на части, например, разделяя строку по символам табуляции.

QFileInfo - Во время поиска возвращает полную информацию о файле.

Класс QDir обеспечивает доступ к структуре каталогов и их содержимого.

QIODevice представляет собой базовый класс всех устройств ввода/вывода в Qt.

Класс QCryptographicHash предоставляет способ генерации криптографических хэшей. QCryptographicHash могут быть использованы для генерации криптографических хэшей двоичных

или текстовых данных.В настоящее время MD4, MD5, и SHA-1 поддерживаются.[6]

QChar обеспечивает поддержку 16-битных символов Unicode.

3.3.1 Автоматизация поиска журнальных файлов

Для сканирования образа на наличие интересующих лог файлов использовался класс QDirIterator. После вызова происходит поочередный обход по каждому файлу в директории и поддиректории. Проверка полученного полного пути к файлу осуществляется регулярным выражением, если условие выполняется, происходит добавление в список обрабатываемых файлов.

3.3.2 Реализация сохранения результатов работы программного комплекса в XML

Сохранение полученных данных происходит в ранее выбранный формат XML(Extensible Markup Language). Для этого используется класс QXmlStreamReader и QxmlStreamWriter. Класс QXmlStreamWriter представляет XML писателя с простым потоковым API.

QXmlStreamWriter работает в связке с QXmlStreamReader для записи XML. Как и связанный класс, он работает с QIODevice, определённым с помощью setDevice().

Сохранение данных реализованно в классе WriteMessage. В методе WriteMessages, структура которого представлена на UML диаграмме в разделе Архитектура.

4 Технические характеристики

4.1 Требования к аппаратному обеспечению

Минимальные системные требования:

– процессор 1ГГц Pentium 4;
– оперативная память 512 Мб;
– место на жёстком диске – 9 Гб.

4.2 Требования к программному обеспечению

Для корректной работы разрабатываемого программного комплекса на компьютере должна быть установлена операционная система Debian Squeeze или выше, данная система должна иметь набор библиотек QT.

4.3 Выбор единого формата выходных файлов

Для вывода результата был выбран формат XML-документов, так как с данным форматом лего работать при помощи программ, а результат работы данного комплекса в дальнейшем планируется обрабатывать при помощи программ.

XML - eXtensible Markup Language или расширяемый язык разметки. Язык XML представляет собой простой и гибкий текстовый формат, подходящий в качестве основы для создания новых языков разметки, которые могут использоваться в публикации документов и обмене данными [7]. Задумка языка в том, что он позволяет дополнять данные метаданными, которые разделяют документ на объекты с атрибутами. Это позволяет упростить программную обработку документов, так как структурирует информацию.

Простейший XML-документ может выглядеть так:

```
<?xml version="1.0"?>
<list_of_items>
<item id="1"\><first/>Первый</item\>
<item id="2"\>Второй <subsub_item\>подпункт
1</subsub_item\></item\>
<item id="3"\>Третий</item\>
<item id="4"\><last/\>Последний</item\>
</list_of_items\>
```

Первая строка - это объявление начала XML-документа, дальше идут элементы документа <list_of_items> - тег описывающий начало элемента
list_of_items, </list_of_items> - тег конца элемента. Между этими тегами заключается описание элемента, которое может содержать текстовую информацию или другие элементы (как в нашем примере). Внутри тега начала элемента так же могут указывать атрибуты элемента, как например атрибут id элемента item, атрибуту должно быть присвоено определенное значение.

5 Разработка программного обеспечения

5.1 Архитектура

5.1.1 Основной алгоритм

В ходе разарботки был применен видоизменнённый шаблон проектирования Factory method.

Данный шаблон относится к классу порождающих шаблонов. Шаблоны данного класса - это шаблоны проектирования, которые абстрагируют процесс инстанцирования (создания экземпляра класса). Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту. Пример организации проекта при использовании шаблона проектирования Factory method представлен на рисунке 5.1.

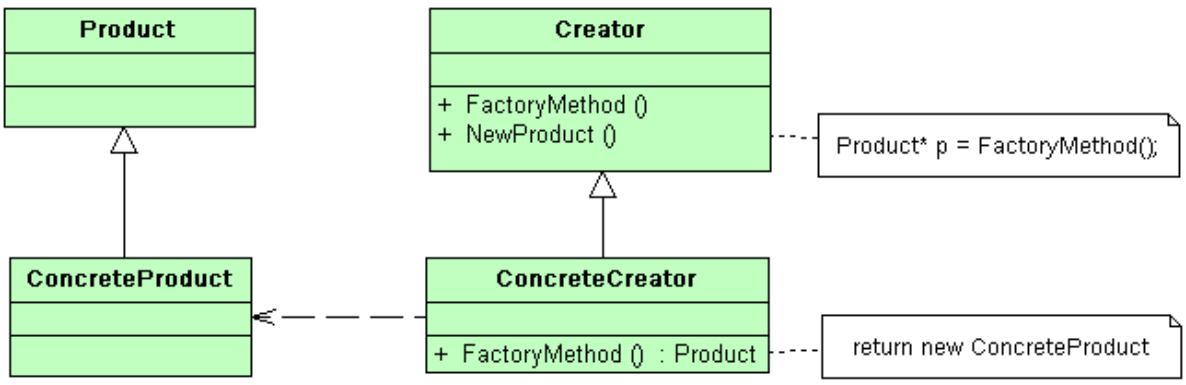

Рисунок 5.1 – Пример организации проекта при использовании шаблона проектирования Factory method

Использование данного шаблона позволило нам разбить наш проект на независимые модули, что весьма упростило задачу разработки, так как написание алгоритма для конкретного таска не влияло на остальную часть проекта. При разработке был реализован базовый класс для работы с образом диска. Данный клас предназначался для формирования списка настроек, определения операционной системы на смонтированном образе и инстанционировании и накапливание всех необходимых классов-тасков в очереди тасков. После чего каждый таск из очереди отправлялся на выполнение. Блоксхема работы алгоритма представлена на рисунке 5.2.

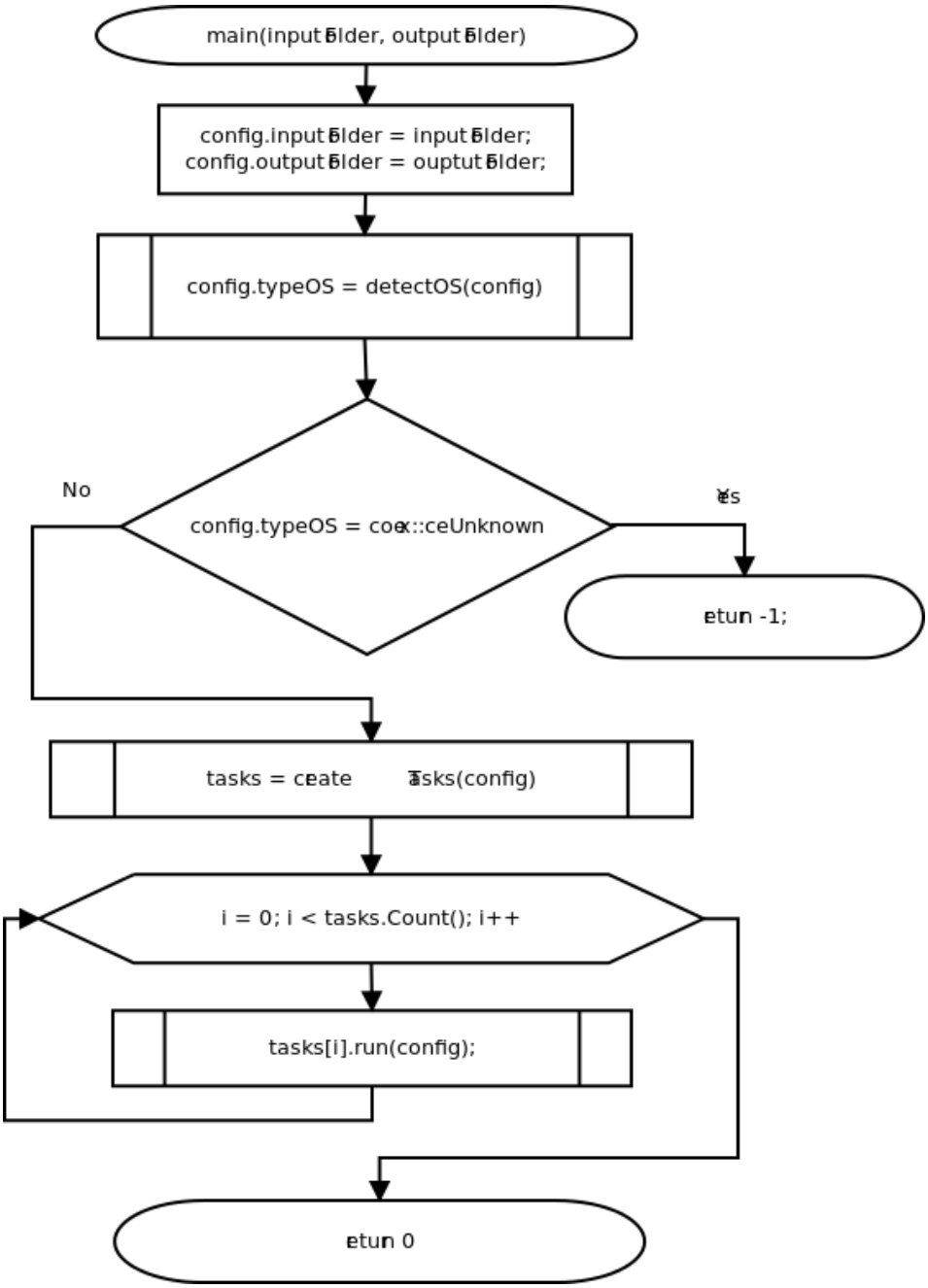

Рисунок 5.2 – Алгоритм работы с образом диска

Каждый класс-таск порождался путем наследования от базового абстрактного класса который имеет 8 методов и 3 атрибута:

1) QString manual() - возвращает справку о входных параметрах данного таска;
2) void setOption(QStringList list) - установка флагов для поданных на вход параметров;
3) QString command() - возвращает команду для инициализации таска вручную;

4) bool supportOS(const coex::typeOS &os) - возвращает флаг, указывающий на возможность использования данного таска для конкретной операционной системы;

5) QString name() - возвращает имя данного таска;

6) QString description() - возвращает краткое описание таска;

7) bool test() - предназначена для теста на доступность таска;

8) bool execute(const coex::config &config) - запуск таска на выполнение;

9) QString m_strName - хранит имя таска;

10) QString m_strDescription - хранит описание таска;

11) bool m_bDebug - флаг для параметра –debug;

На данный момент в проекте используется восемь классов. UML-диаграмма классов представлена на рисунке 5.3.

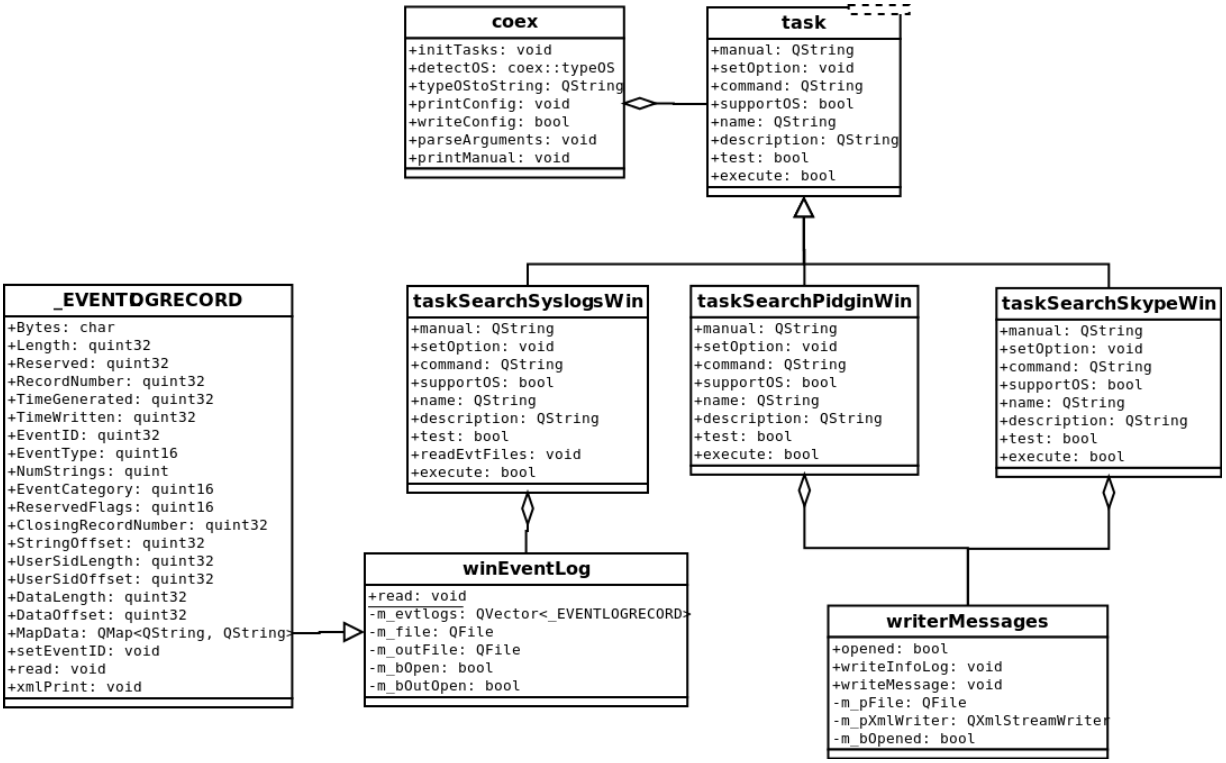

Рисунок 5.3 – UML-диаграмма классов

Классы taskSearchSyslogsWin, taskSearchPidginWin и taskSearchSkypeWin - наследники от класса task являются тасками. Класс winEventLog и _EVENTLOGRECORD предназначины для конвертации журнальных файлов операционной системы Windows XP, а класс writerMessages для преобразования истории переписки.

5.1.2 Описание основных функций модуля системы

Любой модуль системы является классом-наследником от некоторого абстрактного класса используемого как основу для всех модулей программы (шаблон проектирования Factory method). Модуль содержит в себе 8 методов и 3 атрибута:

QString manual() - возвращает справку о входных параметрах данного таска

void setOption(QStringList list) - установка флагов для поданных на вход параметров

QString command() - возвращает команду для инициализации такска вручную

bool supportOS(const coex::typeOS &os) - возвращает флаг указывающий на возможность использования данного таска для конкретной операционной системы

QString name() - возвращает имя данного таска

QString description() - возвращает краткое описание таска

bool test() - предназначена для проверки работоспособности таска

bool execute(const coex::config &config) - запуск таска на выполнение

QString m_strName - хранит имя таска

QString m_strDescription - хранит описание таска

bool m_bDebug - флаг для параметра –debug

5.2 Сбор и анализ системных журналов Windows (III семестр)

Анализ журналов операционных систем может помочь при решении многих задач. Примером таких задач может быть попытка восстановления системы после поломок, поиск причин неполадок системы, просмотр журналов с целью выявления активности приложений в определенные периоды и т.д. и т.п. Также, анализ журнальных файлов является неотъемлемой частью компьютерно-технических экспертиз.

При исследовании компьютеров задача анализа журнальных файлов ставится на этапе «выявление и изучение его роли в рассматриваемом деле», так как именно в журнальных файлах операционной системы хранится информация о действиях производимых на данном компьютере.

Для проведения компьютерных экспертиз существует множество специализированных программных средств, как платных так и свободно распространяемых. Примером такого средства является «OSForensics». Данный инструмент позволяет проводить множество различных исследований таких как просмотр содержимого оперативной памяти, поиск подозрительных файлов на жестком диске, составления списка программ установленных на исследуемом компьютере. Но бесплатная версия не предоставляет никаких средств по работе с журналами операционной системы. Данный факт подтолкнул к созданию собственного программного средства для проведения компьютерных экспертиз.

В данной работе описывается работа по созданию модуля для данного программного средства, позволяющего в автоматическом режиме находить, читать и конвертировать журнальные файлы операционной системы Windows XP в XML-документы, понятные для человека и легко обрабатываемые при помощи
электронно-вычислительных средств.

5.2.1 Общие сведения о журнальных файлах

В операционной системе Windows XP по умолчанию есть четыре журнала:

1) Журнал приложений
2) Журнал безопасности
3) Журнал установки
4) Журнал системы

Каждый из этих журналов содержит определенный тип информации. Журналы приложений содержат информацию о запуске и остановке процессов приложений, изменении статуса каждого

приложения, а также предупреждения и ошибки связанные с приложениями.

Журнал безопасности содержит информацию о входах в систему, и события связанные с безопасностью системы, например превышение количества попыток неправильно введенных подряд паролей.

Журнал установки содержит информацию об установке и обновлении компонентов системы.

Журнал системы хранит информацию о системных событиях. Например изменение схемы энергопотребления или различного рода предупреждения и ошибок.

Каждый журнал хранится в соответствующем файле с расширением .evt. Эти файлы хранятся в папке %windows%/system32/config. Эти файлы имеют специальную структуру.

5.2.2 Структура журнальных файлов операционной системы Windows XP

Все журнальные файлы операционной системы Windows XP имеют единую структуру. Файл представляет собой последовательность записей бинарных данных. Каждая запись – это структура имеющая семнадцать полей [8].

Первые 4 байта содержат длину события в байтах, после длинны идет 4 байтный системный код сообщения. Затем 4 байтовый номер записи, после него дата создания записи, время создания, идентификатор события, тип события и так далее. Ниже приведен список полей записи, предназначенной для считывания одного события из журнального файла.

- uint32 Length;
- uint32 Reserved;
- uint32 RecordNumber;
- uint32 TimeGenerated;
- uint32 TimeWritten;
- uint32 EventID;
- uint16 EventType;
- uint16 NumStrings;
- uint16 EventCategory;
- uint16 ReservedFlags;
- uint32 ClosingRecordNumber;
- uint32 StringOffset;
- uint32 UserSidLength;
- uint32 UserSidOffset;
- uint32 DataLength;
- uint32 DataOffset;
- byte[] Data.

Из сторонних источников [8] стало известно о пяти типах событий (поле EventType) (значение - тип события):

1) 0x0001 - Error event;
2) 0x0010 – Failure Audit event;
3) 0x0008 - Success Audit event;
4) 0x0004 - Information event;

5) 0x0002 - Warning event.

У поля EventID удалось определить четыре значения:

1) 0x00 – Success;
2) 0x01 – Informational;
3) 0x02 – Warning;
4) 0x03 – Error.

Среди множества полей записи события были выделены поля содержащие информацию о типе события, времени возникновения события и создания записи, пользователя от имени которого была сделана запись, а также поле Data - поле с бинарными данными, в которых записана подробная информация о событии.

5.2.3 Алгоритм работы модуля

Модуль выполняет две задачи: поиск журнальных файлов на образе диска и конвертацию каждого файла в XML-документ. Первая задача выполняется при помощи библиотек QDir и QDirIterator из Qt Framework.

QDir — библиотека позволяющая работать с конкретной директорией. Создав объект данного типа с указанием директории мы получим доступ к этой директории в программе и сможем работать в ней (просматривать содержимое; удалять, создавать или копировать файлы; создавать поддиректории). Данный объект так же поддерживать разные наборы фильтров выходных данных которые могут отсеивать ненужную информацию.

QDirIterator — библиотека, предназначенная для работы с файловой системой начиная с определенной директории как точки входа. Создав объект данного типа с указанием директории мы получим все пути которые существуют в файловой системе и начинаются с указанной директории. Данный объект поддерживает фильтрацию которая помогает выделять только необходимую информацию, исключая то, что нас не интересует, например можно вывести список только тех файлов, которые находятся в данной директории или поддиректориях, или исключить вывод символьных ссылок. Объекты данного типа используются для поиска файлов или папок на образе исследуемого диска.

Так же данные библиотеки позволяют создавать объект QFile, который позволяет работать с файлом, путь к которому передается как параметр при создании, данный объект позволяет получить базовую информацию о файле, такую как относительный или абсолютный путь до этого файла, размер файла, тип файла или его имя. Так же позволяет перемещать или копировать данный файл. Поиск работает по алгоритму представленному на рисунке 5.4.

На выходе данного алгоритма мы получаем объект QStringList который содержит пути до всех найденных .evt файлов. Каждый экземпляр коллекции строк данного объекта передается в конструктор объекта winEventLog, который и конвертирует указанный файл в XML-документ. Алгоритм работы конвертора представлен на рисунке 5.5.

Пример результата работы модуля в виде файла в формате XML представлен на рисунке 5.6

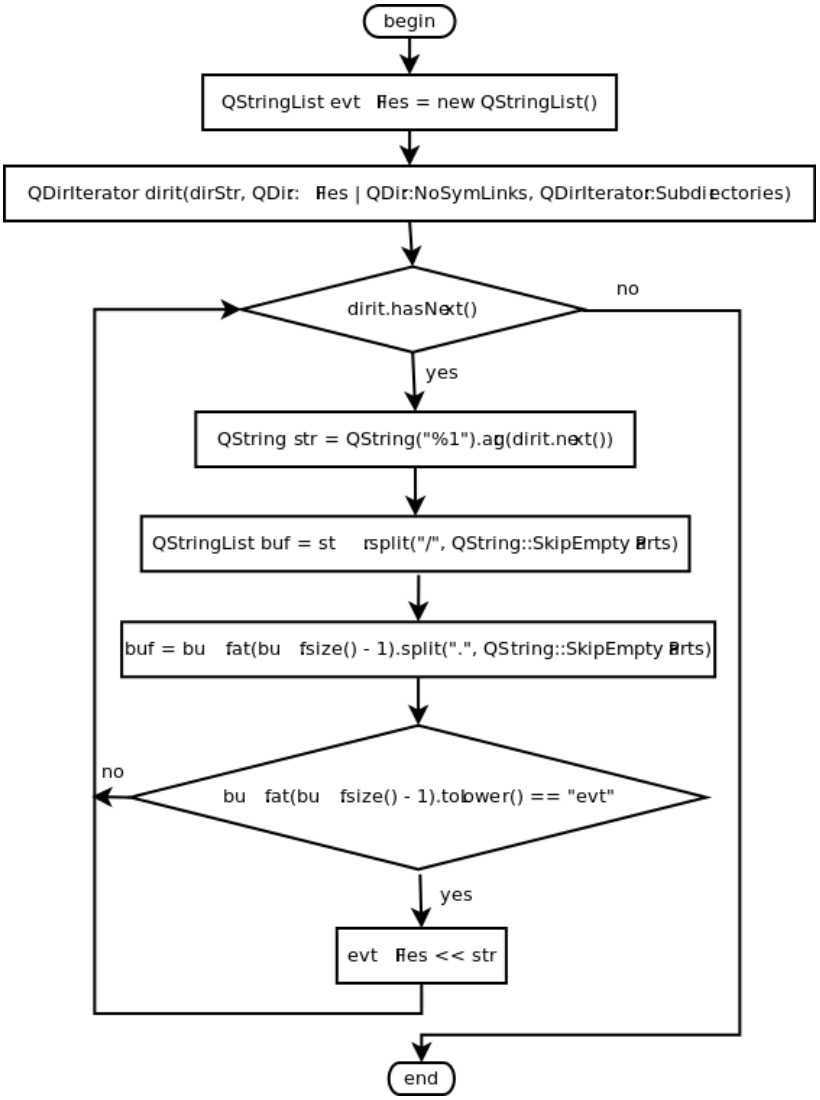

Рисунок 5.4 – Алгоритм поиска файлов с расширением *.evt

5.3 Сбор и анализ системных журналов Windows (IV семестр)

5.3.1 Введение

Продолжая разработку данного программного продукта стало ясно что, несмотря на выбранный шаблон проектирования, добавление нового функционала осложняется тем, что при добавлении новой функции приходится пересобирать весь проект. Это неправльиный подход, так как не позволяет обновлять приложение. Для исправления этой ситуации необходимо доработать архитектуру приложения таким образом, что бы добавление новых функций не приводило к пересборке всего проекта. Это так же позволит проще обновлять проект в дальнейшем.

После доработки архитектуры, необходимо будет переделать проект, переделать реализацию всего фукционала, а так же свести все ветки разработки в одну, так как на данный момент не все изменения применены к проекту.

Сформулируем задачи на семестр:

5.3.2 Задачи на семестр

1) переработать архитектуру проекта для более легкого добавления новых функций

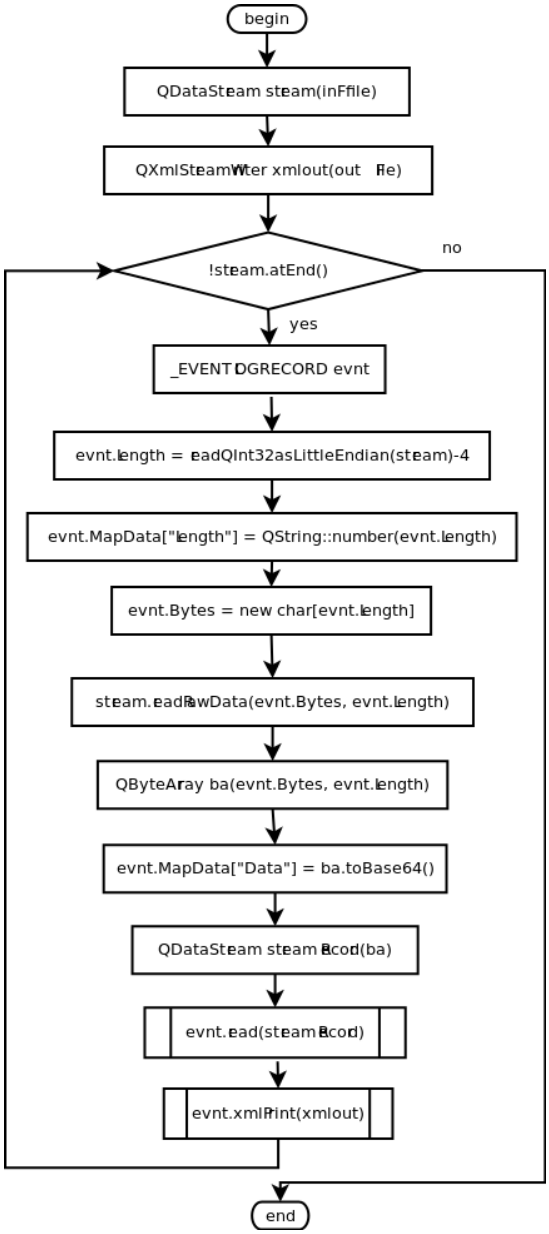

Рисунок 5.5 – Алгоритм конвертирования файлов *.evt в формат XML

2) реализовать данную архитектуру
3) переделать готовые task'и для работы с текущей архитектурой
4) смерджить ветки разработки git

5.3.3 Новая архитектура

С увеличением количества тасков возникла проблема с обновлением кода, так как используемый подход предполагал монолитную программу и все таски хоть и были стандартизированы при помощи шаблона разработки factory, всё равно являлись частью основной программы и для добавления тасков или их обновления необходимо постоянно перекомпилировать весь проект. Данная ситуация крайне неудобна при разработки и тестировании, поэтому необходимо было разделить проект на независимые модули.

В языке программирования C++ есть возможность создания библиотек. Библиотеки – это бинарные файлы в которых хранится коллекции классов и функций. Эти классы и функции можно

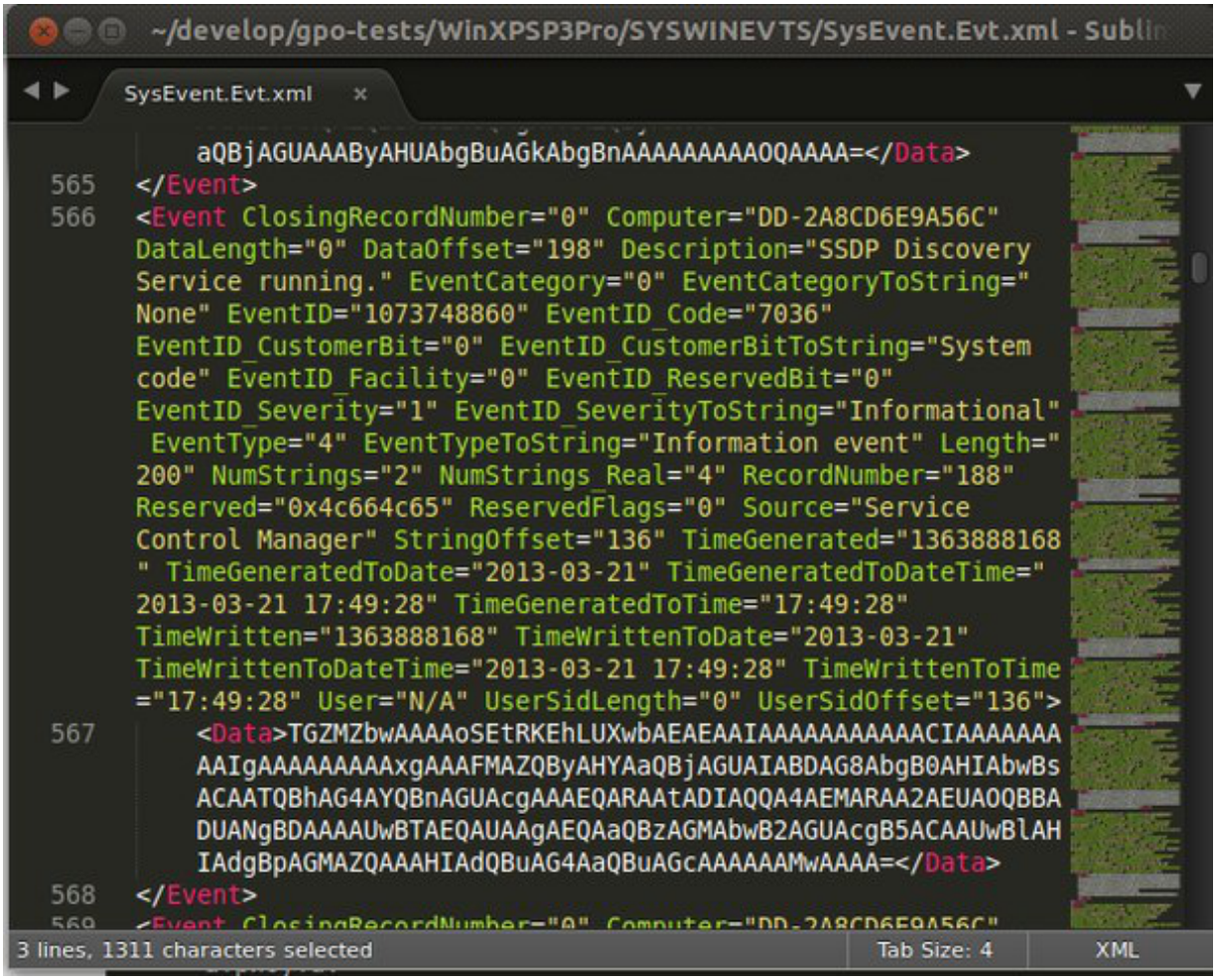

Рисунок 5.6 – Пример лога системы в формате XML

использовать в любой программе в которой данная библиотека подулючается. Подключать библиотеки в программу можно статически (на этапе компиляции программы) или динамически (непосредственно при выполнении программы). У нашего проекта есть «скелет» и таски, каждый из которых работает независимо друг от друга и обменивается данными только со скелетом.

Так как каждая библиотека является самостоятельным файлом с исполняемым кодом, то для его создания и изменения необходим отдельный проект разработки, а значит если мы захотим изменить что-либо внутри библиотеки, то практически наверняка нам не нужно будет перекомпилировать весь проект. Это значительно упрощает операцию обновления программного комплекса. Было принято решение вынести реализацию всех функций скелета в отдельную статическую библиотеку. А все таски оформить в виде динамических библиотек.

Данное решение повлекло за собой ряд проблем:

1) Необходимо разработать структуру динамических библотек для тасков программного комплекса;

2) Необходимо изменить механизм работы с тасками;

3) Необходимо изменить механизм разбора и передачи параметров в основной части программного комплекса;

4) Разработать механизм передачи информации в механизм разбора и передачи параметров.

5.3.4 библиотеки в C++ и их использование

Библиотеки предоставляют программисту возможность использовать при разработке программы готовые фрагменты кода. В библиотеки могут быть включены подпрограммы, структуры данных, классы, макросы. В QT для создания библиотеки необходимо укахать в pro-файле проекта добавить строчку

TEMPLATE = lib

Данная строчка указывает компилятору на то, что мы компилируем библиотеку. После чего в файле исходного кода описывается всё необходимое и после компиляции мы получим файл библиотеки. Для создания динамической библиотеки необходимо добавить в pro-файл ещё строчку:

CONFIG += dll

А так же в заголовочном файле библиотеки поместить описание прототипов функций которые мы хотим подключать динамически в конструкцию

extern «C» «описание прототипов»

Это необходимо для того, что бы при сборке библиотеки функции не поменяли имена. Если не добавить данную конструкцию, то компилятор при сборке изменит имена функций и обратиться к ним динамически не получится.

Для использования разработанной статической библиотеки необходимо просто указать в файле исходного кода строчку

include «abolute/part/to/library/file»

Либо поместить созданную библиотеку в папку с системными библиотеками (или добавить в настройки операционной системы путь до созданной библиотеки), тогда можно подключать библиотеку так же как и стандартные библиотеки (имя библиотеки в треугольных скобках)

Динамические библиотеки используются по-другому. Загрузка функции из динамической библиотеки происходит в три этапа:

1) Загрузка библиотеки;
2) Описание шаблона для вызова функции:
3) Вызов функции.

В качестве примера на рисунке 5.7 представлен пример вызова функции foo, которая принимает на вход один аргумент типа int и возвращает его удвоенное значение. функция находится в библиотеке exampleLib. Загрузка библиотеки производится при помощи средств, предоставляемых Qt framework:

5.3.5 task = динамическая библиотека

Для того, что бы программа могла работать с каждым таском, как с динамической библиотекой, необходимо что бы все библиотеки были созданы по одному шаблону. То есть в каждой библиотеке должен быть определенный набор функций с определенным именем и входными параметрами к которым можно обращаться при загрузки библиотеки.

В каждой библиотеке необходимо определить две функции:

1) функция, возвращающая информацию о таске (имя таска и возможный набор параметров, предназначенных для данного таска)

2) функция выполнения данного таска, которая возвращает логическое значение по оконча-

```
#include <QLibrary>
#include <iostream>

int main()
{
    //данная строка загружает динамическую библиотеку, которая лежит
    //по указанному пути. ссылка на библиотеку хранится в переменной lib
    QLibrary lib("/home/user/mylibs/exampleLib");
    //описание переменной. на экран выведится "4"
    int a = 4;
    std::cout << a << std::endl;
    //описываем тип данных для функции, которую хотим загрузить из библиотеки
    //первым делом указываем типо возвращаемого значения, после чего имя
    //указателя на данный тип и в скобках перечисляем типы данных аргументов
    typedef int (*Fct)(int);
    //загружаем функцию с именем foo из библиотеки
    Fct fct = (Fct)(lib.resolve("foo"));
    //использование загруженной функции
    a = fct(a);
    //в консоли выведется 8
    std::cout << a << std::endl;
    return 0;
}
```

Рисунок 5.7 – Пример вызова функции foo

нию работы, на вход она принимает переменную со всеми настройками, введенными пользователем

На данный момент механизм разбора флагов не был реализван. Структура библиотеки представлена на рисунке 5.8.

Все реализованные таски были оформлены в виде динамических библиотек. Алгоритмы обработки данных изменению не подвергались.

5.3.6 новый механизм работы с тасками

Для того, что бы было возмодно добавлять в наш комплекс новые таски без перекомпиляции скелета комплекса мы оформили их в виде динамических библиотек, каждая из которых реализована по шаблону, рассмотренному выше. Все библиотеки, которые мы хотим загружать при работе программы мы помещаем в специальную дирректорию libs рядом с исполняемым файлом. При запуске программа просматривает данную дирректорию и поочередно загружает каждую библиотеку и запускает её на выполнение (если это возможно).

Алгоритм работы данного механизма представлен на рисунке 5.9

5.3.7 мерджим ветки git

Разработка комплекса велась двумя разработчиками, каждый из которых работал в своей ветке разработки системы контроля версий git. Необходимо было собрать все изменнеия проекта воедино. Обычно для этого используются инструменты, предстовавляемые самим gitom, но так как в проекте была изменена архитектура (по сути был создан новый проект, который делали их частей старого), то было принято решение просто получить все изменнеия с каждой ветки для каждого про-

```
#include <coex>
#include <QString>


//описание прототипов функций, которые будут вызывать при загрузке библиотеки
extern "C"
{
    //получение информации о библиотеке
    QString getLibInfo();
    //выполнение задачи, решаемой данной библиотекой
    bool execute(const coex::config &);
}

//блок со всеми ресурсами, необходимыми библиотеке для решения задачи
//все что описано в этом блоке доступно только внутри библиотеки
void supportFun1();
int supportFun1(int, int);

class supportClass
{
public:
    supportClass();
    ~supportClass();

    //class description
}
```

Рисунок 5.8 – Структура библиотеки

екта и оформить последнии версии тасков в виде динамических библиотек. Данный подход повзолил избежать процедуры разрешения конфликтов. На данный момент новая версия проекта находится в ветке master и содержит все последнии изменения предыдущего проекта. Разработка первой версии комплекса остановлена и, после того как весь код старого проекта будет использован, данная версия комплекса будет удалена.

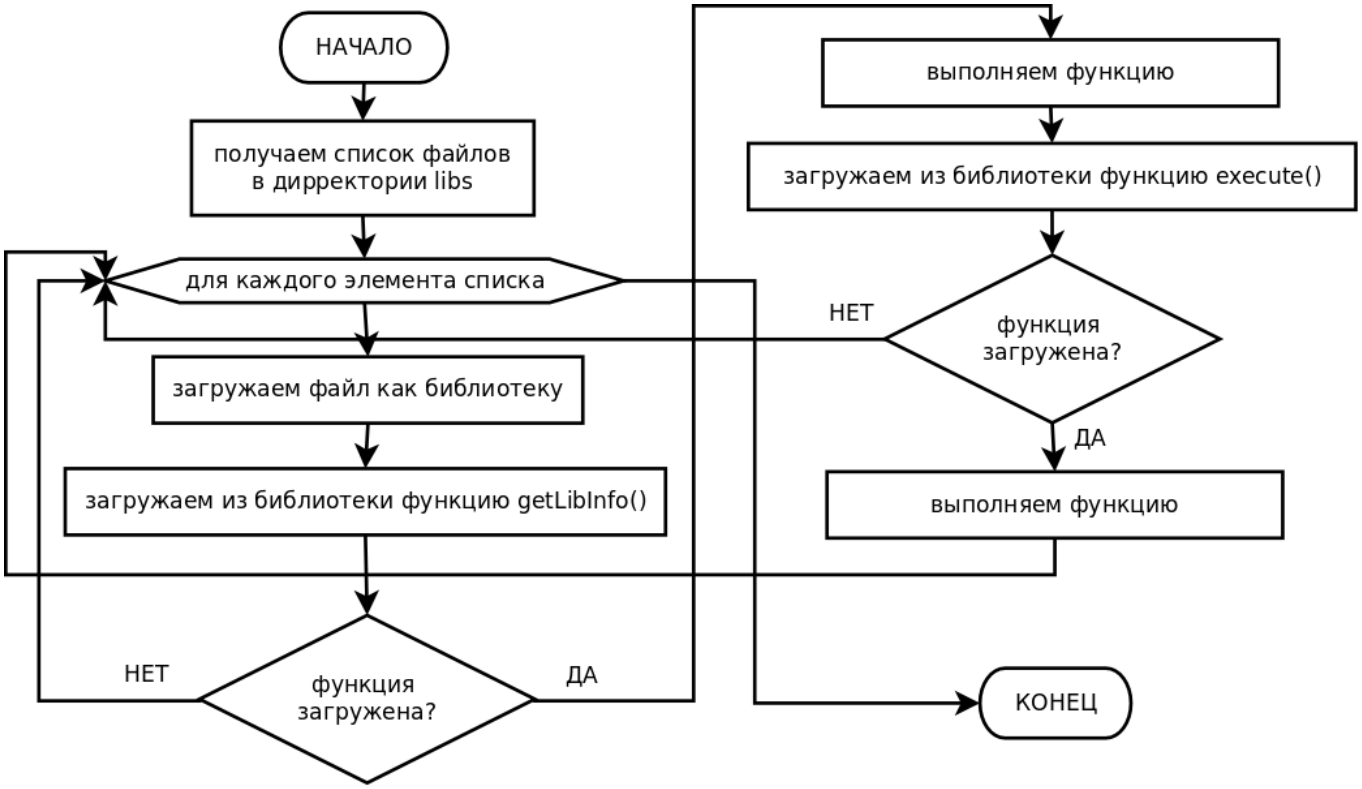

Рисунок 5.9 – Пример вызова функции foo

5.4 Сбор и анализ истории переписки мессенджеров (III семестр)

Опишем задачу более подробно, разделив её на подзадачи:

1) определить места хранения файлов с личной перепиской пользователя;
2) определить формат файлов переписки;
3) разбор найденных файлов;
4) автоматизировать процесс поиска файлов;
5) производить сохранение полученных информации формат XML.

5.4.1 Определение местоположения логов переписки мессенджера Skype

По умолчанию файлы располагаются в директориях, для разных операционных систем возможны незначительные изменения (таблица 5.1).

Таблица 5.1 – Директории хранения логов Skype

Windows Vista/7/8	C:\Users\username\ AppData\Roaming\Skype
Windows XP	C:\Documents and Settings\username\ Application Data\Skype
Linux(Ubuntu)	/home/username/.Skype

Основная интересующая нас информация находится в main.db [9]. Алгоритм поиска логов представлен на рисунке 5.10.

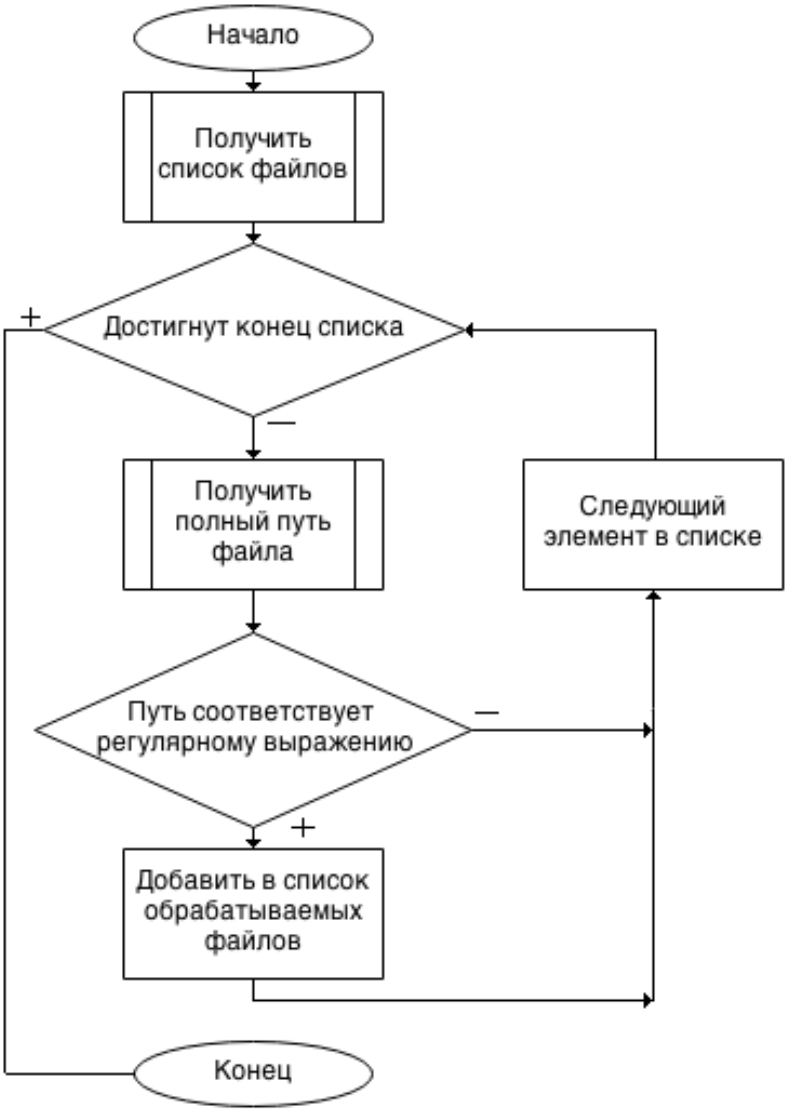

Рисунок 5.10 – Блок-схема алгоритма поиска логов

5.4.2 Определение местоположения логов переписки мессенджера Pidgin

Определение месторасположения файлов переписки происходит следующим образом: для примонтированного образа запускается модуль, который сужает область поиска, сканируя только нужные директории образа.

По умолчанию файлы располагаются в директориях, для разных операционных систем возможны незначительные изменения (таблица 5.2).

Таблица 5.2 – Директории хранения логов pidgin

Windows Vista/7/8	C:\Users\username\AppData\ Roaming\.purple\logs
Windows XP	C:\Documents and Settings\username\ Application Data\.purple
Windows 98/ME	C:\Windows\Profiles\username\.purple
Linux(Ubuntu)	/home/username/.purple/logs

Основная интересующая нас информация хранится в файлах с маской имени YEAR-MONTH-DATE.TIME.html(например 2013-03-02.004915+0700NOVT.htm).

5.4.3 Формат хранения переписки мессенджера Skype

Приложение «skype» хранит переписку на локальных машинах пользователей или же возможна синхронизация с машин других пользователей [9]. Формат хранения: реляционная база данных основная на СУБД SQLite. Но база данных «main.db» не является единственным местом хранения информации, Skype сохраняет сведения о работе программы во временных файлах (chatsync). Эти файлы имеют расширение «.dat» и цифробуквенные имена (например «0172b0a519e2c584»)[10].

5.4.4 Формат хранения переписки мессенджера Pidgin

Приложение «pidgin» поддерживает перечисленные ниже протоколы: [11]

– ”AIM”
– ”Bonjour”
– ”Gadu-Gadu
– ”Google Talk”
– ”Groupwise”
– ”ICQ”
– ”IRC”
– ”MSN”
– ”MXit”
– ”MySpaceIM”
– ”SILC”
– ”SIMPLE”
– ”Sametime”
– ”XMPP”
– ”Yahoo!”
– ”Zephyr”

Соответственно, все подлюченные аккаунты всех вышеперечисленных протоколов хранятся как лог файлы на локальной машине пользователя в формате HTML и TXT. По умолчанию лог файлы хранятся в .HTML файле. Настройки программы, пользователя и подключенных аккаунтов хранятся в XML.

Примечание: Кроме account.xml - он хранит нешифрованные пароли для всех подключенных чатов [11].

5.4.5 Алгоритм работы модуля Skype

Структура рассматриваемого файла.

main.db содержит 18 таблиц. и ещё немного статистики.

– ”DbMeta”
– ”Contacts”
– ”LegacyMessages”

– ”Calls”
– ”Accounts”
– ”Transfers”
– ”Voicemails”
– ”Chats”
– ”Messages”
– ”ContactGroups”
– ”Videos”
– ”SMSes”
– ”CallMembers”
– ”ChatMembers”
– ”Alerts”
– ”Conversations”
– ”Participants”
– ”VideoMessages”

Таблицы которые были рассмотрены, на данный момент:

1) Contacts
2) Messages
3) Chats
4) Calls
5) CallMembers
6) Conversations

В таблице Contacts находятся все контакты, причем даже те, что были удалены, и уже не показываются в клиенте.

```
select skypename, fullname, languages, country, city
from contacts
```

В таблицах Calls и CallMembers содержатся, соответственно, история звонков и их участников.

```
select calls.id as ”ID разговора”, coalesce(contacts.displ
accounts.fullname) as ”Инициатор”, strftime(’%d.%m.%Y
%H:%M:%S’,calls.begin_timestamp, ’unixepoch’,
’localtime’) as ”Дата начала”, time(calls.duration,
’unixepoch’) as ”Длительность”, callmembers.dispname
as ”Подключенный участник”, strftime(’%d.%m.%Y
%H:%M:%S’,callmembers.start_timestamp, ’unixepoch’,
’localtime’) as ”Дата подключения”, time(callmembers.call_
’unixepoch’) as ”Длительность подключения”
from calls inner join callmembers on calls.id
= callmembers.call_db_id left join contacts on
calls.host_identity = contacts.skypename left join
accounts on calls.host_identity = accounts.skypename
```

И, наконец, в таблицах Conversations и Messages содержатся данные переписки и сами сооб-

щения.

```
select conversations.id as "ID переписки",
conversations.displayname as "Участники перепис-
ки", messages.from_dispname as "Автор сообщения",
strftime('%d.%m.%Y %H:%M:%S',messages.timestamp,
'unixepoch', 'localtime') as "Время сообщения",
messages.body_xml as "Текст сообщения" from
conversations inner join messages on conversations.id
= messages.convo_id order by messages.timestamp
```

Для доступа ко всему содержимому базы достаточно иметь доступ к самому файлу — содержимое базы никак не шифруется и не защищается, так что любой человек, который сможет получить доступ к вашему профилю Windows, сможет найти список контактов, просмотреть историю звонков и прочитать всю переписку.

5.4.6 Алгоритм работы модуля Pidgin

Структура рассматриваемого файла.

У каждого лог файла есть заголовок находящийся между тегов title. В котором записан ID_Chat, дата начала переписки, логин пользователя и используемый протокол.

Затем идет "тело" в котором описывается обмен сообщениями в формате, время, автор сообщения и сообщение.

Пример "Заголовка" `<head><meta http-equiv = "content-type" content= "text/html; charset=UTF-8"> <title> Conversation with 0dpkhcz6clufs2kozj82uqif30@public.talk.g at Чт. 24 окт. 2013 23:40:21 on user.fox@gmail.com/ (jabber)</title></head>`

Пример "Тела" `<font color="#16569E"> <font size="2"> (23:44:52) </font> <b>user.fox@gmail.com/95C9F047: </b></font> Hello) <br/>`

Точно так же из полученного списка найденные файлы поочередно открываются для чтения. Разбор открытого файла решено осуществлять при помощи регулярных выражений, описанных в класс QRegExp.

Парсинг логов pidgin представлен на рисунке 5.11.

5.4.7 Структура логов Skype, сохранённых в XML

Все документы XML начинаются с пролога (prolog). Пролог сообщает, что документ написан на XML, а также указывает, какая версия XML при этом использовалась.

Следующий элемент Messages с атрибутом Messenger, ему присваивается имя рассматриваемого приложения.

Для каждого файла БД, находящейся в обрабатываемом списке, создается элемент info account содержащий атрибуты skypeName, fullName, emails, ipcountry которым присваивается, соответственно: логин пользователя Skype, его полное имя, email (не шифрованный), геолокация ip-

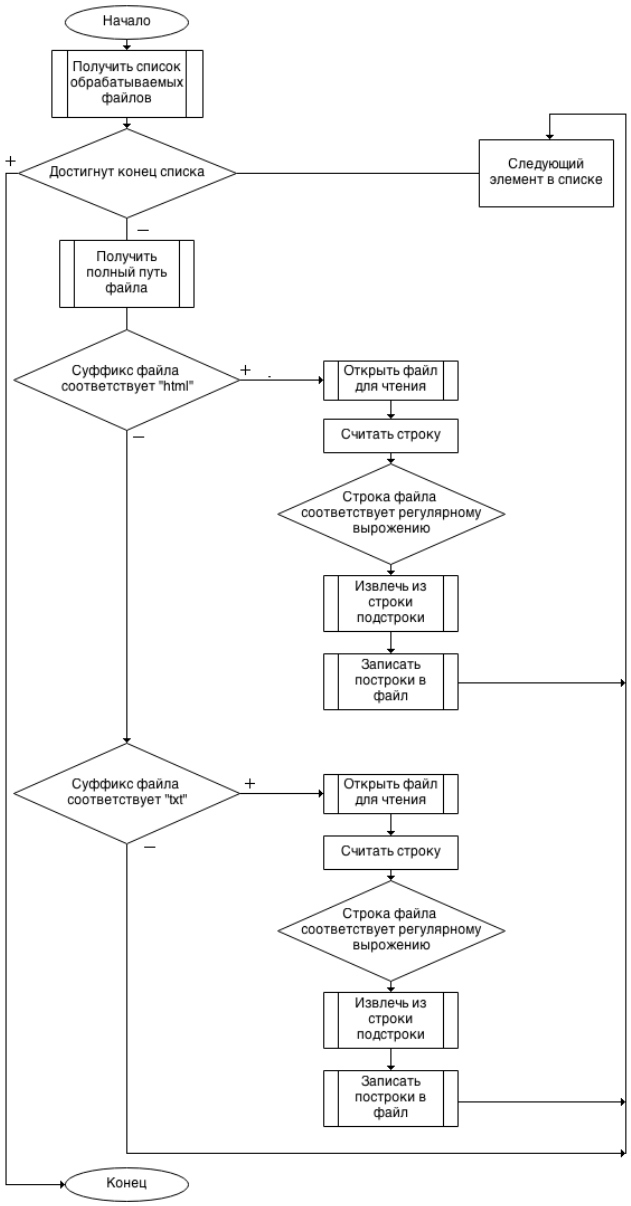

Рисунок 5.11 – Блок-схема алгоритма парсинга логов

адреса.

Далее следует элемент contact содержащий атрибуты skypeName, fullName, languages, country, city.

Пример выходного файла в формате XML представлен на рисунке 5.12.

5.4.8 Структура логов Pidgin, сохранённых в XML

Все документы XML начинаются с пролога (prolog). Пролог сообщает, что документ написан на XML, а также указывает, какая версия XML при этом использовалась.

Следующий элемент Messages с атрибутом Messenger, которому присваивается имя рассматриваемого приложения.

Далее следует элемент INFO содержащий атрибуты chathID, account, data, protocol которым присваивается, соответственно: идентификатор чата, полная дата в формате День.Число Месяц. Год Час:Мин:Сек, аккаунт с которого происходил обмен сообщениями и протокол используемый для передачи сообщений.

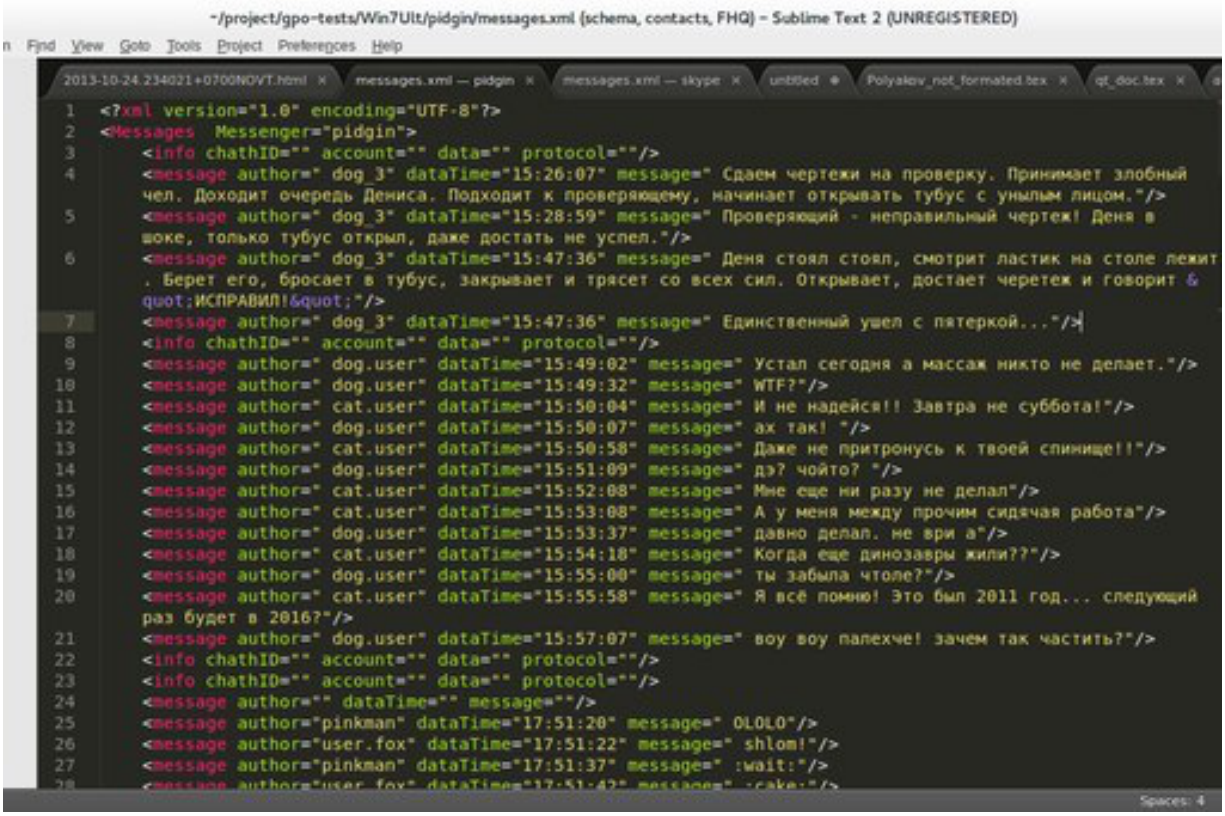

Рисунок 5.12 – логи Skype в формате XML

Последний элемент – MESSAGE, содержащий атрибуты author, dataTime, message.

Пример выходного файла в формате XML представлен на рисунке 5.13.

На текущий момент полностью реализован импорт контактной книги из приложения Skype, импорт звонков и переписки находится в режиме разработки.

5.5 Сбор и анализ истории переписки мессенджеров (IV семестр)

5.5.1 Сбор и анализ истории переписки мессенджеров

1) преобразование собранной информации в читаемый вид;
2) добавление новых функций к уже имеющимся.

Для упрощения разобьем задачу, на подзадачи:

1) Создание веток (branch), и распределенная работа программистов;
2) Визуализация данных;
3) Выбор и описание утилит для преобразования;
4) XSTL;
5) Показать результат;
6) Функция чтения контактной книги pidgin;
7) Функция сбора используемых учётных записей pidgin;
8) Описание работы SAX и DOM модели разбора XML;
9) Функция сбора дополнительных данных skype.

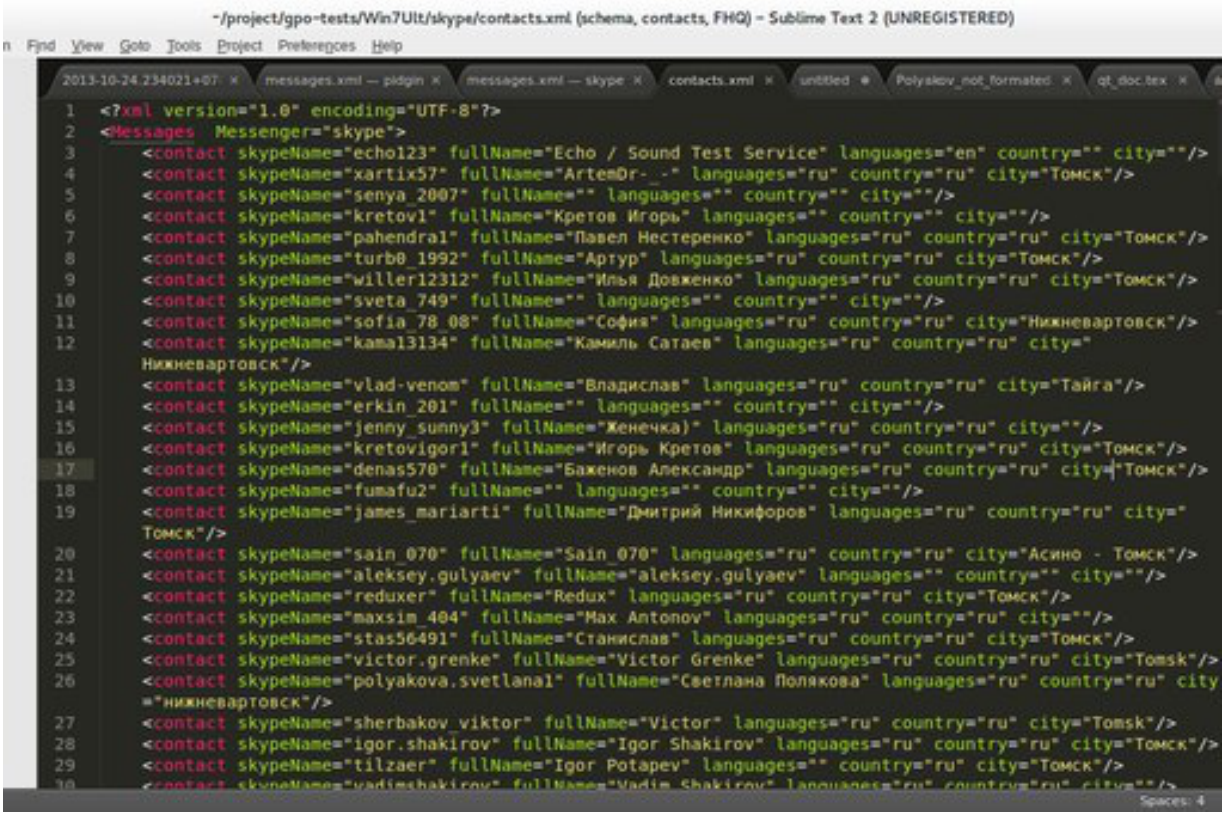

Рисунок 5.13 – логи Pidgin в формате XML

5.5.2 Создание веток (branch), и распределенная работа программистов.

Допустим мы приступаем к задаче, которая займет больше одного дня\недели. Разработка ПО в комманде подразумевает частое создание коммитов. Это позволяет чаще получать изменения и избегать кофликтов.

Но если задача длинная, и мы будем коммитить полусделанную задачу, то это будет как минимум мешать другим разработчикам, как максимум сделает проект неработоспособным.[2]

Однако не делать коммиты несколько дней тоже плохой вариант. Вот для этого и существуют другие ветки. Они позволяют нам вносить изменения, не мешаю и этом остальным разработчикам. Или же работать над несколькими ветками одновременно.

Ветка - это снимок репозитария в сделанный в прошлом, в который не попадают коммиты из основной ветки после момента создания.

В проекте использовались две ветки:

1) master
2) skype_to_XML

В ветке «master» велась работа по переносу проекта с модульной структуры на плагииную. “skype_to_XML” велась работа по расширению функционала модулей клиентов - мгновенного обмена сообщениями, и VoIP программ. Велась работа по ознакомлению с языком разметки xslt, с последующей генерацией отчета в xhtml, основанный, на собранных данных, хранящихся в xml.

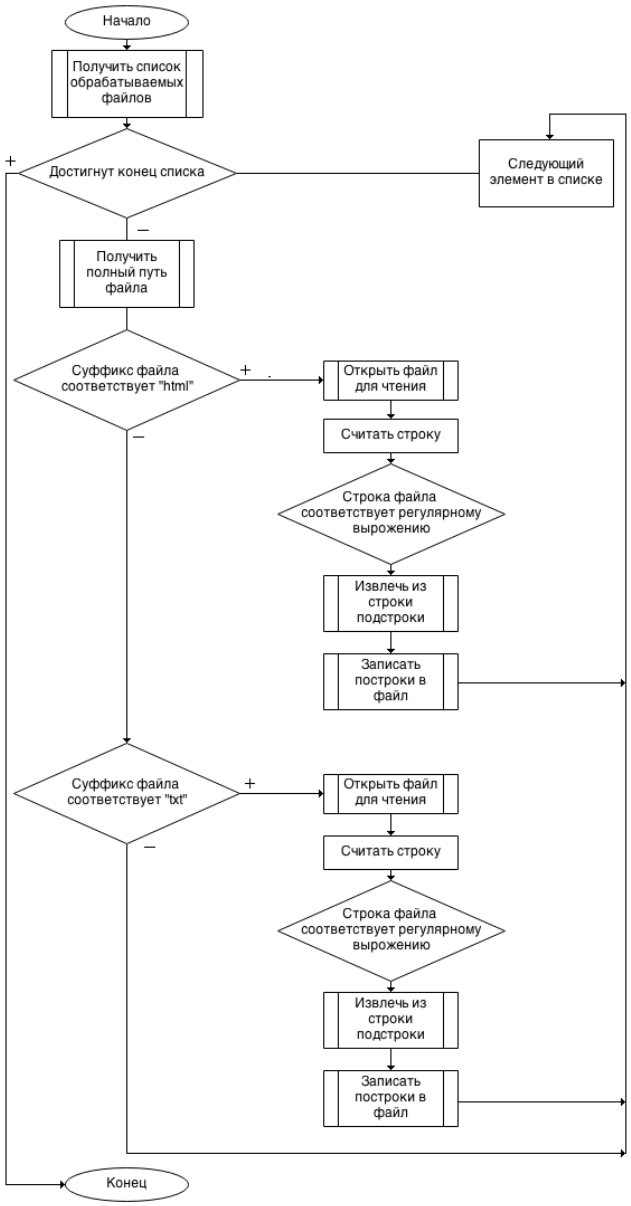

Рисунок 5.14 – Результат работы git branch

5.6 Визуализация данных

Визуализация собранной информации очень важный момент создания подобного рода комплексов, т.к. это является главной задачей нашего ПО.

Список возможных вариантов конвертации XML

1) CSV
2) HTML
3) TXT
4) SQL
5) PNG
6) И многое другое...

Из рассматриваемого списка удобным для восприятия, хранения структурированной информации и програмно-независимымоказался - HTML.

5.7 Выбор и описание утилит для преобразования

XML можно хранить виде БД, а затем выводить записи БД в виде веб-страниц, используя к примеру PHP. Возникает вопрос, зачем мы храним данные в промежуточном формате XML, если в результате данные хранятся в БД. Работа в таком формате займет большее время, и сложнее ко всему. Для прямого преобразования XML в XHTML, мы можем воспользоваться языком преобразования XSLT.

5.8 XSTL

XSLT (eXtensible Stylesheet Language Transformations) — это расширяемый язык стилей для преобразований, который использует для описания преобразований структуры документов.[12]

Хотя XSLT не позиционируется, как язык запросов для XML, можно смело сравнить есго с языком SQL, в котором определяются запросы к реляционным базам данных.[12]

При применении таблицы стилей XSLT, состоящей из набора шаблонов, к XML-документу образуется конечное дерево, которое может быть сериализовано в виде XML-документа, XHTML-документа (только для XSLT 2.0), HTML-документа или простого текстового файла. Правила выбора (и, отчасти, преобразования) данных из исходного дерева пишутся на языке запросов XPath. XSLT имеет множество различных применений, в основном в области веб-программирования и генерации отчётов. Одной из задач, решаемых языком XSLT, является отделение данных от их представления, как часть общей парадигмы MVC (англ. Model-view-controller). Другой стандартной задачей является преобразование XML-документов из одной XML-схемы в другую.[12]

5.9 Результаты работы

5.10 Описание работы SAX и DOM модели разбора XML

SAX расшифровывается как Simple API for XML, что означает буквально «Простой прикладной интерфейс программирования для XML»

SAX (англ. «Simple API for XML») — способ последовательного чтения/записи XML-файлов.

Всё, что делает SAX-парсер, это сообщает вызвавшему приложению о встреченных распознанных элементах XML-разметки или о встреченных ошибках. Связь парсера с вызывающим приложением, как правило, осуществляется посредством функций обратного вызова.

DOM (от англ. Document Object Model — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.[12]

5.11 функция чтения используемых аккаунтов и контактной книги pidgin

Интересующие нас контактная книга находятся в в файле blist.xml, используемые аккаунты хранятся в файле accounts.xml.

Оба файла имеют идентичную древовидную структуру, которая начинается в “корне” и разветвляется до “листьев”, как и положено в XML документе.

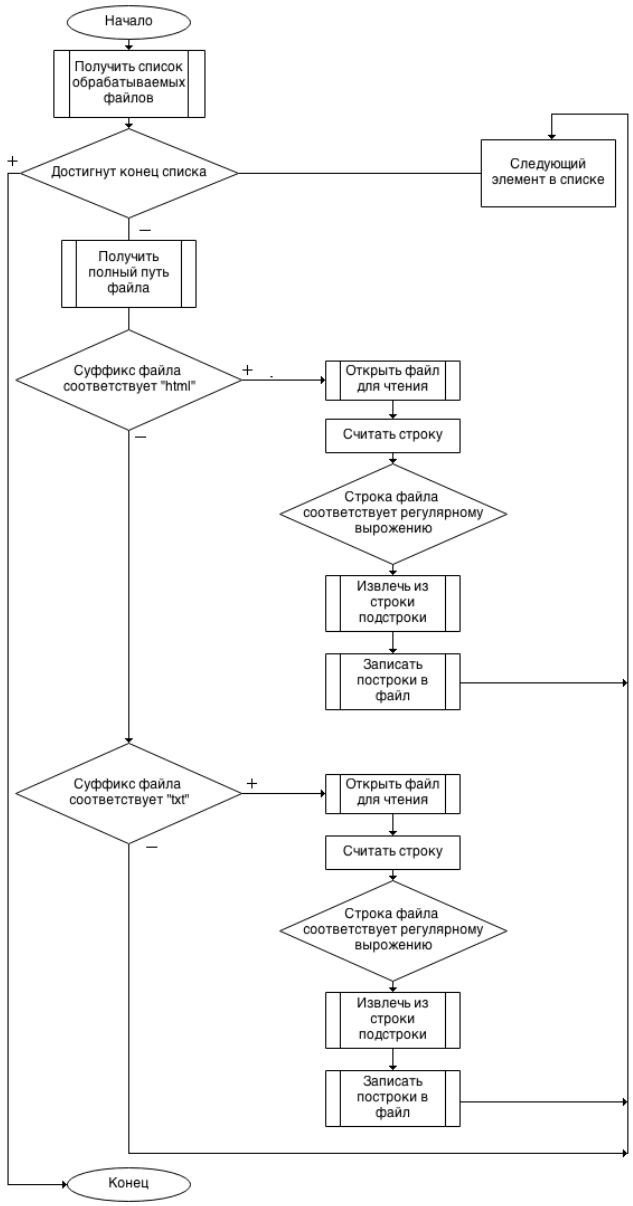

Рисунок 5.15 – Результат заглавной генерированной страницы отчета

Для разбора данного файла использовалась модель SAX. Реализованная в библиотеке QT, в классе QXml. При проходе по дереву XML мы реагируем на события [5]:

– Старт документа
– Старт элемента
– CDATA (анг. Character Data) Символьный тип данных
– Комментарии
– Конец документа
– Конец элемента
– Ошибка чтения

При проходе парсера по файлу мы реагируем на нужные нам элементы, атрибуты и прочее. Список интересующих элементов файла acoumt.xml

– элемент protocol - используемый протокол для передачи сообщений.
– элемент name Имя или другое идентифицирующий элемент.
– элемент password Пароль.

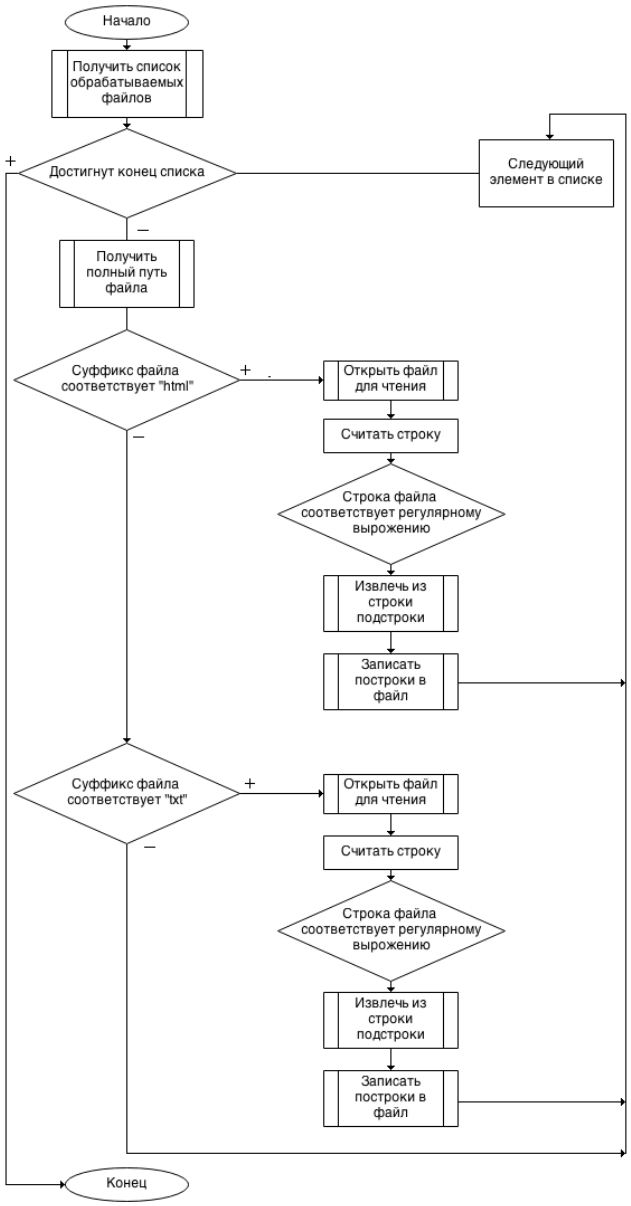

Рисунок 5.16 – Результат сбора данных контактной книги pidgin

– элемент alias Имя используемое при переписке.

Список интересующих элементов файла blist.xml

– элемент buddy: атрибут account - используемый акаунт.

– элемент buddy: атрибут proto - используемый протокол для передачи сообщений.

– элемент setting: атрибут last_seen - здесь хранится время последнего появления контакта в сети интернет. Даные хранятся в UnixTime.

– элемент alias - Имя контакта используемое при переписке.

Затем данные вычитываются, и сохраняются в XML отчет.

5.12 функция сбора дополнительных данных skype

К имеющемуся списку функций работы с логами VoIP skype в этом семестре добавилось. Функция извлечения информации о аккаунтах, когда-либо авторизовавшихся на данном компьютере. Т.е при авторизации на компьютере создается база данных формата SQLite main.db. Одна из таблиц acount содержит в себе информацию о используемом\подключенном акаунте.

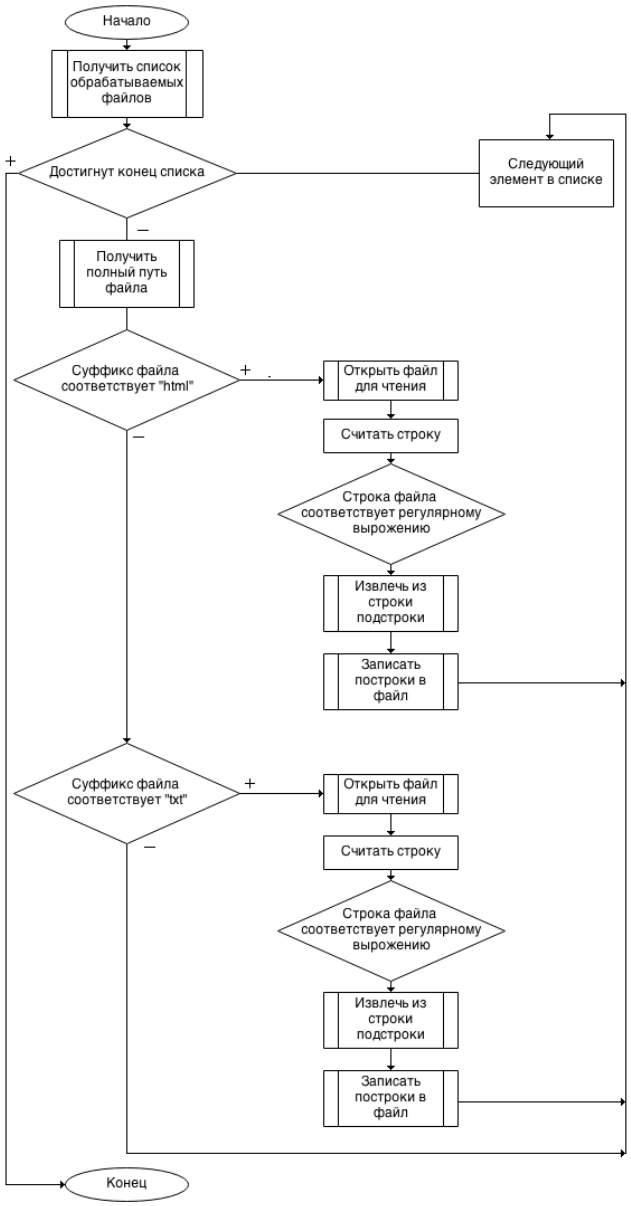

Рисунок 5.17 – Результат сбора данных аккаунта pidgin

– полное имя
– логин skype
– почта
– хеш-сумма используемого пароля
– язык
– данные о геолокации(Страна, город, часовой пояс)

Функция извлечения информации о звонках,входящих или исходящих когда-либо производившихся на данном компьютере. Таблица calls содержит в себе информацию данного типа.

– пользователь совершающий звонок
– пользователь принимающий звонок
– время звонка

Функция извлечения переписки между пользователями skype. Таблица chats содержит в себе информацию данного типа.

– пользователь передающий сообщение

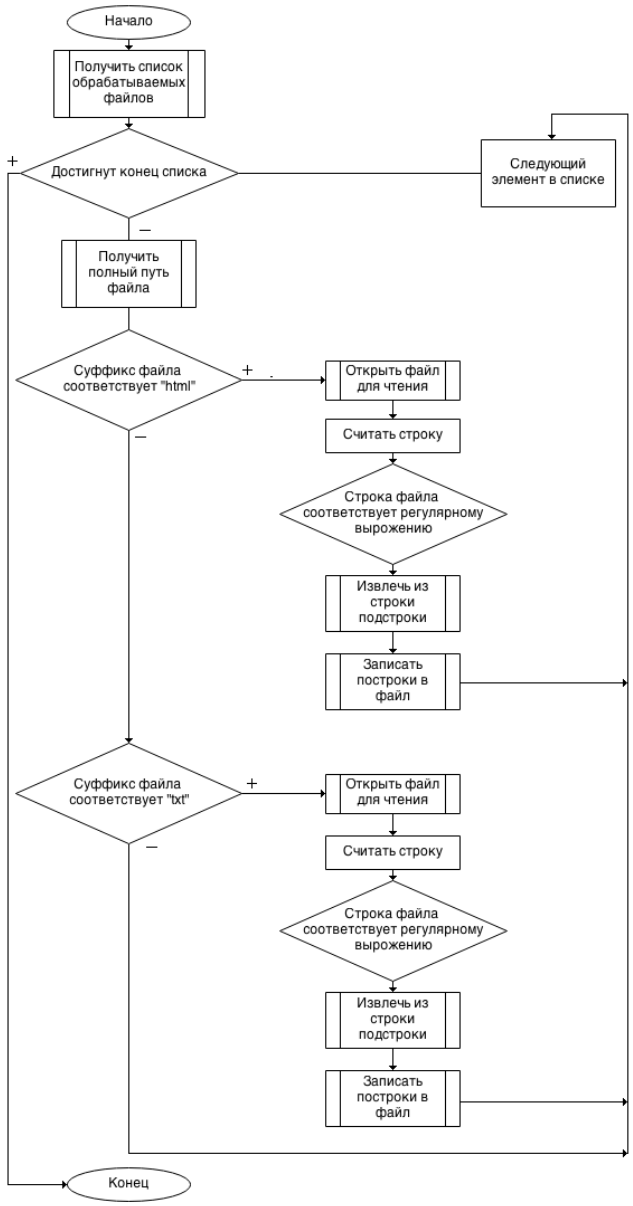

Рисунок 5.18 – Результат сбора данных контактной книги skype

– пользователь принимающий сообщение
– время создания сообщение

Данные вычитываются посредством SQL запросов, реализованных через библиотеку Qt, в классе QSql*, и затем так же сохраняются в XML отчет.

6 Некоторые аспекты сертификации программных средств объектов информатизации по требованиям информационной безопасности

Под сертификацией принято понимать независимое подтверждение соответствия тех или иных характеристик оборудования или информационной системы некоторым требованиям. В нашем случае речь идет о программном комплексе для проведения компьютерной экспертизы, во время экспертизы программный комплекс хранит и обрабатывает информацию, которая может относиться к персональным данным, коммерческой или государственной тайне и др. а также на основании результатов работы программы специалист проводящий экспертизу пишет заключение, которое может быть использовано как доказательство в суде. Поэтому к данной системе могут быть применены

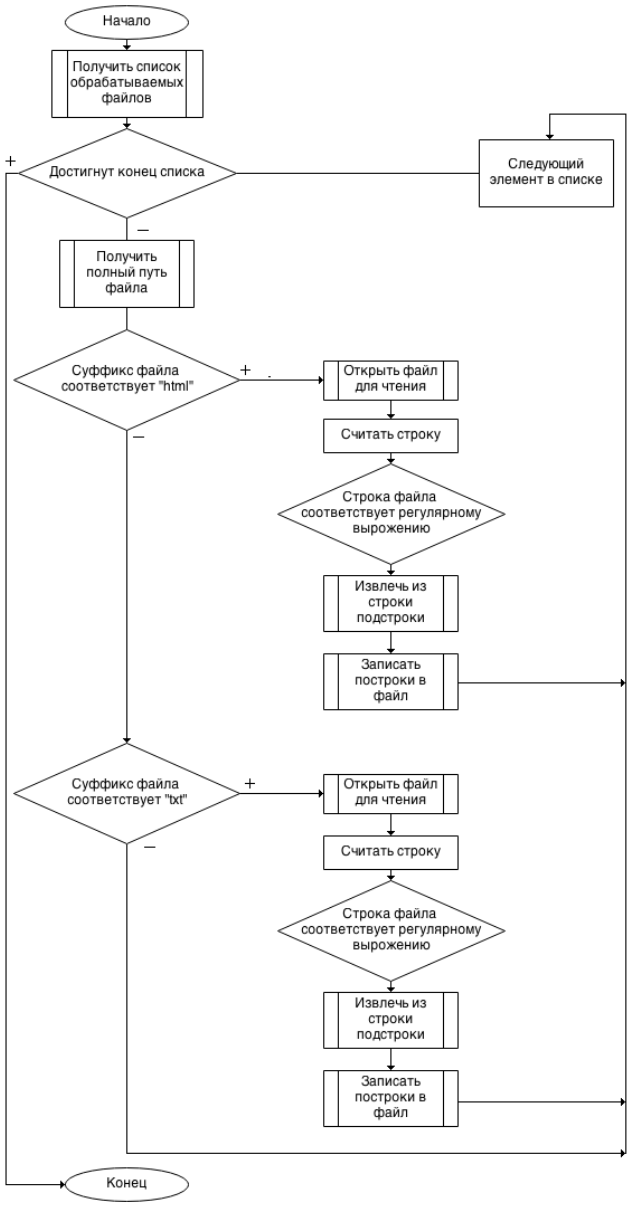

Рисунок 5.19 – Результат сбора данных аккаунта skype

требования, касающаяся безопасности информации.

Для обеспечения высокого показателя качества необходима сертификация технологии и системы обеспечения качества их проектирования, разработки и сопровождения.

Проведение сертификации системы обычно планируется для достижения одной или нескольких целей:

– определения соответствия или несоответствия элементов системы установленным требованиям;

– определения эффективности внедренной системы с точки зрения соответствия поставленным целям для обеспечения качества продукции;

– обеспечения возможности улучшить свою систему;

– определения соответствия системы регламентирующим требованиям.

Первоначальные затраты на их проведение могут нести инициаторы испытаний, заказчик, а также ее разработчики и поставщик.

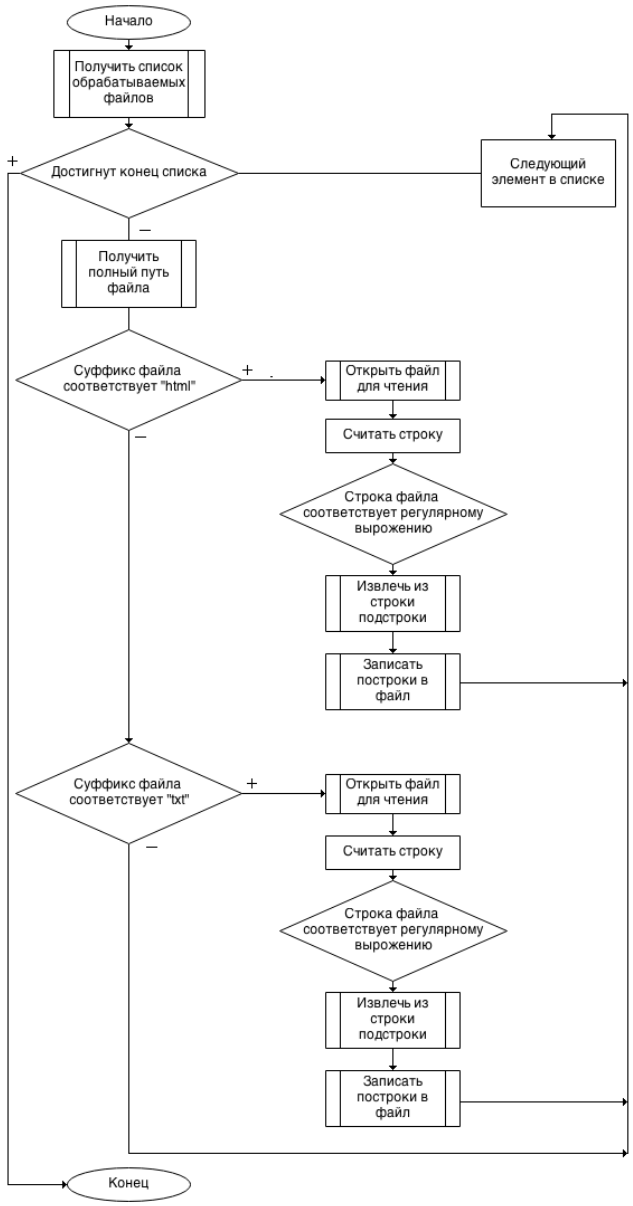

Рисунок 5.20 – Результат сбора данных звонков skype

6.1 Схема проведения сертификации

Особенность любых сертификационных испытаний — это независимость испытательной лаборатории сертифицирующей организации, осуществляющей независимый контроль результатов испытаний.

Схема проведения сертификации:

1) Заявитель (в нашем случае разработчик) подает в федеральный орган (ФСБ, ФСТЭК или Минобороны) заявку на проведение сертификационных испытаний.

2) Федеральный орган определяет аккредитованную испытательную лабораторию и орган по сертификации.

3) Испытательная лаборатория совместно с заявителем проводит сертификационные испытания. Если в процессе испытаний выявляются те или иные несоответствия заявленным требованиям, то они могут быть устранены заявителем в рабочем порядке, либо может быть принято решение об изменении требований к продукту, например о снижении класса защищенности.

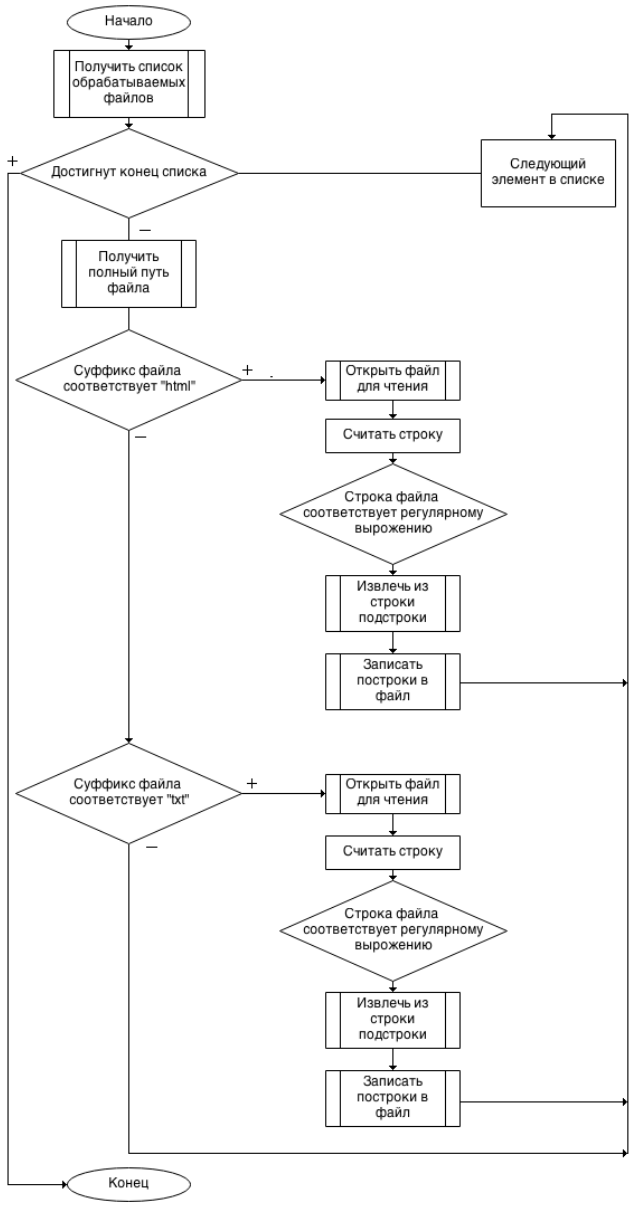

Рисунок 5.21 – Результат сбора установленного ПО

4) Материалы испытаний передаются в орган по сертификации, который проводит их независимую экспертизу. В экспертизе участвуют не менее двух независимых экспертов.

5) Федеральный орган по сертификации на основании заключения органа по сертификации оформляет сертификат соответствия. При каких-либо несоответствиях, федеральный орган может провести дополнительную экспертизу с привлечением экспертов из различных аккредитованных лабораторий и органов.

В случае возникновения инцидентов на объектах информатизации, связанных с утечкой информации, регулирующие органы могут проинспектировать лабораторию, которая проводила испытания, и приостановить лицензию и аттестаты аккредитации.

Деятельность российских систем сертификации в РФ регламентируется Федеральным законом № 184 «О техническом регулировании». Сертификация средств защиты информации может быть добровольной или обязательной — проводимой главным образом в рамках Минобороны, ФСБ и ФСТЭК. К компетенции ФСБ относятся средства защиты информации, применяемые в органах государственной власти. Система сертификации средств защиты информации Минобороны, в свою оче-

редь, ориентирована на программные изделия, применяемые на объектах военного назначения.[13]

6.2 Добровольная сертификация

Добровольная сертификация применяется с целью повышения конкурентоспособности продукции, расширения сферы ее использования и получения дополнительных экономических преимуществ. Результаты сертификации должны оправдывать затраты на ее проведение вследствие получения пользователями продукции с показателями более высокого качества. Таким сертификационным испытаниям подвергаются компоненты операционных систем и пакеты прикладных программ широкого применения. Добровольные системы сертификации средств защиты информации на сегодняшний день пока еще не получили широкого распространения. Такого рода системой является «АйТи-Сертифика», но более известным средством сертификации является ЦЕРБЕР. К сожалению, несмотря на то что в добровольных системах можно получить сертификат на соответствие любому нормативному документу по защите конфиденциальной информации, при аттестации объектов информатизации такие сертификаты ФСТЭК России не признаются.

6.3 Обязательная сертификация

Обязательная сертификация необходима для программных продуктов и их производства, выполняющих особо ответственные функции, в которых недостаточное качество, ошибки или отказы могут нанести большой ущерб или опасны для жизни и здоровья людей. Примером такой сертификации служат программные продукты в авиации, в атомной энергетике, в военных системах и т.д.

Существуют два основных подхода к сертификации (таблица 6.3):

Таблица 6.3 – Основные подходы к сертификации

Подход	Документы
Функциональное тестирование	в соответствии с положениями стандарта ГОСТ Р 15408
Структурное тестирование	по ГОСТ Р 51275-99

Функциональное тестирование средств защиты информации, позволяющее убедиться в том, что продукт действительно реализует заявленные функции. Это тестирование проводится в соответствии с определенными нормативными документами Гостехкомиссии России. Если же не существует документа, которому сертифицируемый продукт соответствовал бы в полной мере, то функциональные требования могут быть сформулированы в явном виде – в технических условиях, или в виде задания по безопасности.

Структурное тестирование программного кода на отсутствие недекларированных возможностей. Недекларированными возможностями,например, являются программные закладки, которые при возникновении определенных условий позволяют осуществлять несанкционированные воздействия на информацию. Выявление недекларированных возможностей - проведение серии тестов исходных текстов программ.

В большинстве случаев средство защиты информации должно быть сертифицировано как в части основного функционала, так и на предмет отсутствия недекларированных возможностей. Делается исключение для систем обработки персональных данных второго и третьего класса с целью

снижения затрат на защиту информации для небольших частных организаций. Если программное средство не имеет каких-либо механизмов защиты информации, оно может быть сертифицировано только на предмет отсутствия недекларированных возможностей. [14]

6.4 Список организаций, проводящих сертификацию

Так как в некотором будущем в нашей архитектуре должен появиться модуль системы защиты данных от НСД рассмотрим несколько организаций, которые могут провести сертификацию нашего программного комплекса (таблица 6.4):

Таблица 6.4 – Организации проводящие сертификацию

Организация	Описание	Сертификация продуктов
Радиофизические тестовые технологии	Данный орган по сертификации програмных средств аккредитован Ростехрегулированием(Госстандартом) в системе сертификации ГОСТ Р.	1 операционные системы и средства их расширения, 2 системы программирования и утилиты, 3 сетевое программное обеспечение, 4 системы управления базами данных (СУБД), 5 электронные таблицы, 6 базы данных и информационно-справочные системы, 7 электронные архивы. 8 и другие.
АНО МИЦ	Центр зарегистрировал в Росстандарте систему добровольной сертификации программного обеспечения и аппаратно-программных комплексов	1 ПО, используемое для моделирования технологических процессов, математического и иного моделирования; 2 ПО передачи, хранения, актуализации, защиты, обеспечения доступа и использования измерительной, вычислительной и иной информации; 3 ПО баз данных; 4 и другие.

Агенство РСТ	уполномоченный агент Регионального органа по сертификации и тестированию РОСТЕСТ-МОСКВА, одного из крупнейших сертификационных и испытательных центров в Европе. Сертификаты РОСТЕСТ-МОСКВА пользуются особым доверием потребителей и контролирующих организаций.	1 электронные базы данных и информационные системы; 2 системы управления базами данных; 3 сервисные программы (утилиты); 4 ПО для организации и работы с сетью; 5 и другие.
ЗАО «НПО «Эшелон»	аккредитовано в качестве органа по сертификации ФСТЭК России и испытательных лабораторий в системах обязательной сертификации Минобороны России, ФСБ России, ФСТЭК России и добровольной сертификации «АйТи-Сертифика».	1 соответствие требованиям по защищенности от несанкционированного доступа к информации; 2 отсутствие недекларированных возможностей (в т.ч. программных закладок), 3 соответствие заданию безопасности (по требованиям ГОСТ Р ИСО/МЭК 15408); 4 соответствие реальных и декларируемых в документации функциональных возможностей (в т. числе и на соответствие требованиям Технических Условий); 5 и другие.
Центр сертификации «Мостест»	Структурное подразделение органа по сертификации «Гортест», предоставляет полный комплекс услуг, связанных с подготовкой и оформлением различных разрешительных документов.	1 автоматизированных систем, в т.ч. проектирования (САПР) и управления различными технологическими процессами (АСУ ТП), систем управления отраслями и объединениями, ПО для технологической подготовки производства; 2 систем программирования, утилит, операционных систем;

		3 сетевого ПО и приложений мультимедиа-и т.д.; 4 систем управления базами данных, баз данных, информационных и справочных систем; 5 электронных изданий, архивов, таблиц; 6 и другое.

В данном разделе отчета была исследована информация о сертификации программных средств и др. Также были найдены организации, которые могут произвести сертификацию нашего программного комплекса. Это необходимо для возможности использовать данных комплекс для работы с информацией различного уровня конфиденциальности. Что в свою очередь даст возможность использовать данный программный комплекс в государственных и коммерческих учреждениях для проведения компьютерной экспертизы. Также сертификация обеспечит высокий показатель качества программного комплекса для выхода на рынок.

7 Создание собственного репозитория для debian и ubuntu

С развитием разрабатываемого программного комплекса возникла необходимость организовать свой репозиторий. Свой репозиторий нужен для того, чтобы выложить программное обеспечение в открытый доступ в удобном для установки и обновления виде.

7.1 Порядок конфигурирования репозитория

Для начала следует выбрать дирректорию, в которой мы хотим выкладывать наши deb пакеты. Пускай это будет /var/www/repositories/debian.

```
prefix=/var/www/repositories/debian
mkdir -p \$prefix/main
```

Копируем все необходимые deb пакеты в /var/www/repositories/debian/main, там можно устроить любую структуру каталогов. Сделаем подкаталоги для каждого пакета и положим в них файлы .deb, .dsc, .changes и .tar.gz. Далее переходим в корень нашего репозитория.

```
cd \$prefix
```

Создаем индексные файлы. Для бинарных пакетов сделать это можно с помощью команды dpkg-scanpackages.

```
dir=\$prefix/dists/unstable/main/binary-i386\\
mkdir -p \$dir\\
dpkg-scanpackages --arch i386 main /dev/null > \$dir/Packages\\
gzip -9c <\$dir/Packages >\$dir/Packages.gz\\
bzip2 -9c <\$dir/Packages >\$dir/Packages.bz2
```

Проделаем это для каждой архитектуры, для которой есть пакеты в main. Затем необходимо создать индексные файлы для исходных текстов

```
dir=\$prefix/dists/unstable/main/source\\
mkdir -p \$dir\\
dpkg-scansources main /dev/null > \$dir/Sources\\
gzip -9c <\$dir/Sources >\$dir/Sources.gz\\
bzip2 -9c <\$dir/Sources >\$dir/Sources.bz2
```

И наконец, необходимо создать файл $prefix/dists/unstable/main/Release, который имеет следующую структуру:

```
Archive: unstable
Suite: unstable
Component: main
Origin:
Label: Debian
Architecture:
```

Создадим такой файл и заполним необходимые поля, затем дадим команды

```
dir=\$prefix/dists/unstable/main
apt-ftparchive release \$dir >>\$dir/Release
```

Последняя команда вычисляет MD5Sum, SHA1 и SHA256 для файлов Packages* и Sources*.

Теперь осталось только подписать репозиторий. Сделаем это следующей командой:

```
gpg -abs -o \$dir/Release.gpg \$dir/Release
```

Для того, чтобы aptitude, apt и др. смогли найти новый репозиторий, добавим необходимые записи в /etc/apt/sources.list.

```
deb http://example.com/repository/debian unstable main
deb-src http://example.com/repository/debian unstable main
```

8 Поисковая система Apache Solr

8.1 Общая информация об Apache Lucene и Solr

The Apache Lucene – это свободная библиотека для высокоскоростного полнотекстового поиска, написанная на Java. Может быть использована для поиска в интернете и других областях компьютерной лингвистики (аналитическая философия).

Основные возможности:

1) Масштабируемая и высокоскоростная индексация
   свыше 95GB в час на современном оборудовании
   требуется малый объем RAM — «heap» всего 1MB
   размер индекса примерно 20-30 % от размера исходного текста
2) Мощный, точный и эффективный поисковый алгоритм
   ранжированный поиск — лучшие результаты показываются первыми

множество мощных типов запросов: запрос фразы, wildcard запросы, поиск интервалов и т. д.

поиск основанный на «полях» (таких как, заголовок, автор, текст)

возможность сортировать по различным полям

multiple-index поиск с возможностью объединения результатов

возможность одновременного поиска и обновления индекса

3) Кроссплатформное решение

исходный код полностью написан на Java

наличие портов на другие языки программирования

Apache Solr – открытая корпоративная поисковая платформа, созданная на базе Apache Lucene. Основные возможности Apache Solr включают в себя мощный полнотекстовый поиск, подсветку результатов поиска, фасетный поиск, динамическую кластеризацию результатов, интеграцию с СУБД и поддержку индексирования документов разных форматов (например, MS Word и PDF). Также имеется возможность создания распределенной поисковой системы и репликация поискового индекса.

Проект Apache Solr написан на Java и запускается как самостоятельное приложение в контейнере сервлетов, например, на Apache Tomcat. Solr использует Lucene как поисковый механизм и имеет REST HTTP/XML и JSON интерфейс для взаимодействия с другими приложениями.

8.2 Установка Apache Solr

Существует 3 готовых сборки Apache Solr из репозитория из них 2 для различных веб-серверов и 1 исходный код сервиса написанного на java. Сначала необходимо установить необходимый минимум програмных пакетов, которые позволят работать с Solr.

Как говорилось ранее поисковой сервис написан на языке java, поэтому в первую очередь необходимо установить JavaJDK, для этого открываем консоль и добавляем по очереди строки следующего содержания:

```
apt-get update
apt-get install sun-java7-jdk
```

После установки JavaJDK скачиваем с официального сайта http://lucene.apache.org/solr/ исходники с поисковым сервисом актуальной версии, после чего разархивируем выполнив команду:

```
tar -xzvf ./solr-4.10.2.tgz
```

Для запуска сервиса используетя скрипт написанный на java, чтобы его запустить необходимо перейти в папку example, которая находится ~/solr-4.10.2/example/, и запустить скрипт. Пример как это проделать приведен ниже:

```
cd ~/solr-4.10.2/example/
java -jar start.jar
```

Запустив сервис мы можем открыть его веб оболочку набрав в браузере http://localhost:8983/solr (рисунок 8.1).

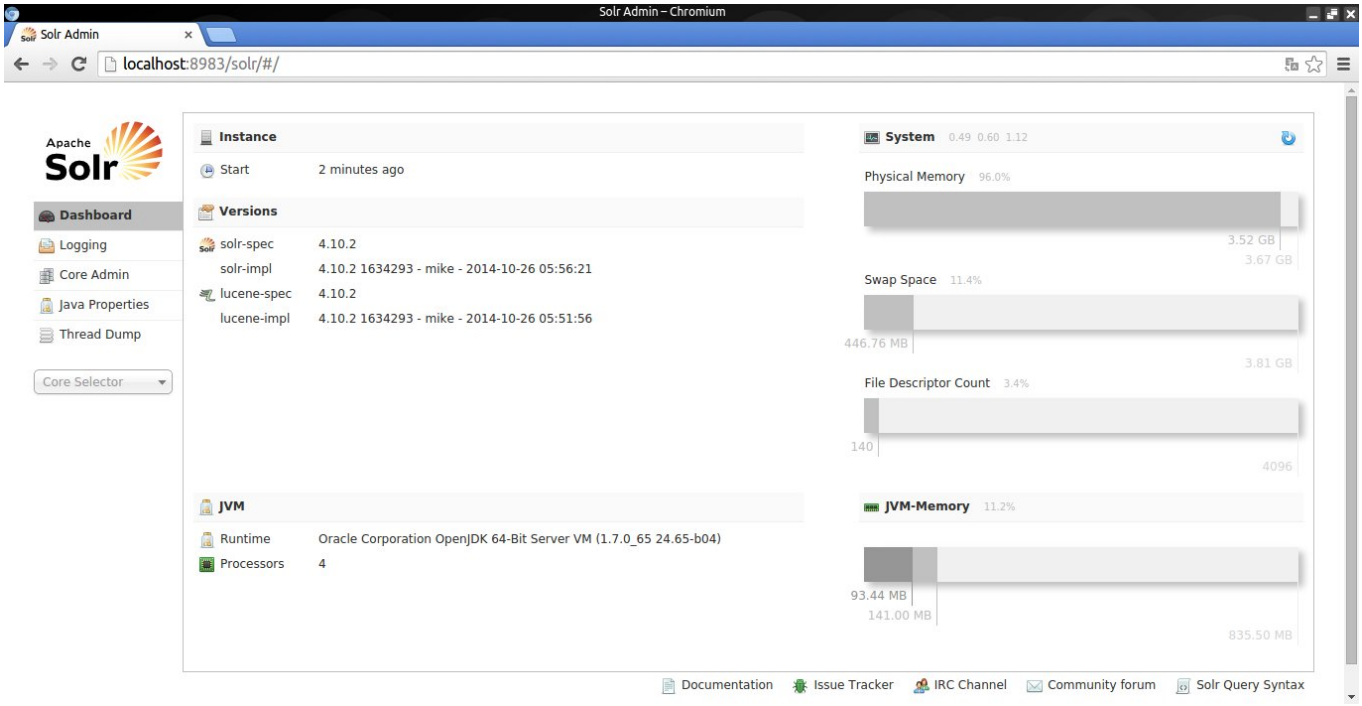

Рисунок 8.1 – Веб-оболочка Solr

8.3 Написание конфигурационного файла schema.xml

Для того, чтобы Solr искал по тем полям, которые нам нужны, необходимо его правильно сконфигурировать. Поля, используемые при индексировании и поиске информации по индексу, описываются в файле «schema.xml».

Большую часть содержимого файла мы оставили такой, какой она была по умолчанию, заменив только строки, описывающие поля для индексирования.

Были добавлены следующие строки:

```
<!-- browser's fields -->
   <field name="history_url" type="text" indexed="true" stored="true"/>
   <field name="history_date" type="text" indexed="true" stored="true"/>
   <field name="bookmark_url" type="text" indexed="true" stored="true"/>
   <field name="bookmark_name" type="text" indexed="true" stored="true"/>
   <field name="bookmark_other" type="text" indexed="true" stored="true"/>
   <field name="login_data_url" type="text" indexed="true" stored="true"/>
   <field name="login_data_name" type="text" indexed="true" stored="true"/>
   <field name="login_data_password" type="text" indexed="true" stored="true"/>
   <field name="preferences_param_name" type="text" indexed="true"
        stored="true"/>
   <field name="preferences_param_value" type="text" indexed="true"
        stored="true"/>

<!-- Message's fields -->
   <field name="log_chat_id" type="text" indexed="true" stored="true"/>
   <field name="account_name" type="text" indexed="true" stored="true"/>
   <field name="account_fullName" type="text" indexed="true" stored="true"/>
```

```
<field name="account_emails" type="text" indexed="true" stored="true"/>
<field name="account_ipcountry" type="text" indexed="true" stored="true"/>
<field name="call_timestamp" type="text" indexed="true" stored="true"/>
<field name="call_duration" type="text" indexed="true" stored="true"/>
<field name="call_server" type="text" indexed="true" stored="true"/>
<field name="call_client" type="text" indexed="true" stored="true"/>
<field name="contact_name" type="text" indexed="true" stored="true"/>
<field name="contact_fullName" type="text" indexed="true" stored="true"/>
<field name="contact_birthday" type="text" indexed="true" stored="true"/>
<field name="contact_phone" type="text" indexed="true" stored="true"/>
<field name="contact_gender" type="text" indexed="true" stored="true"/>
<field name="contact_languages" type="text" indexed="true" stored="true"/>
<field name="contact_country" type="text" indexed="true" stored="true"/>
<field name="contact_city" type="text" indexed="true" stored="true"/>
<field name="message_chat_id" type="text" indexed="true" stored="true"/>
<field name="message_account" type="text" indexed="true" stored="true"/>
<field name="message_protocol" type="text" indexed="true" stored="true"/>
<field name="message_dataTime" type="text" indexed="true" stored="true"/>
<field name="message_message" type="text" indexed="true" stored="true"/>
<field name="message_author" type="text" indexed="true" stored="true"/>
<field name="message_text" type="text" indexed="true" stored="true"/>
<field name="contact_account" type="text" indexed="true" stored="true"/>
<field name="contact_protocol" type="text" indexed="true" stored="true"/>
<field name="contact_id" type="text" indexed="true" stored="true"/>
<field name="account_id" type="text" indexed="true" stored="true"/>
<field name="account_mail" type="text" indexed="true" stored="true"/>
<field name="account_protocol" type="text" indexed="true" stored="true"/>
<field name="account_password" type="text" indexed="true" stored="true"/>
```

9 TaskSearchArchive

9.1 Структура RAR архива

RAR — проприетарный формат сжатия данных и условно-бесплатная программа-архиватор. Имеет расширение .rar, .rev, .r00 или .r01

Определение архивного файла RAR

RAR-архив состоит из блоков переменной длины с заголовками по 7 байт каждый. Любой архив содержит как минимум два блока MARK_HEAD и MAIN_HEAD. Первый содержит информацию о том, что перед нами RAR, и выглядит как “0x 52 61 72 21 1A 07 00” в HEX’ах, или 0x 52 45 7E 5E” в старых версиях . Третий байт 0x72 как раз таки указывает на то, что это Marker Header. Слово 00 07 в little-endian содержит длину блока. Как раз таки 7 байт.[**?**]

Второй блок Main Header начинается сразу же после первого и должен содержать 13 байт и иметь маркировочный байт равным 0x73. После него в файле уже начинаются данные — будь-то сжатый файл (маркет 0x74 в третьем байте заголовка блока), комментарий к архиву, дополнительная

информация или, к примеру, recovery-запись.[**?**]

9.1.1 Формат архивного файла RAR

Файл архива состоит из блоков разной длины.

В самом начале архива стоит блок-маркер, после которого идёт блок заголовка архива, за которым в произвольном порядке следуют блоки остальных типов.

Каждый блок начинается со следующих полей (таблица 9.5):

Таблица 9.5 – Поля блоков архива

HEAD_CRC	2 байта	CRC всего блока или его части
HEAD_TYPE	1 байт	Тип блока
HEAD_FLAGS	16 бит (=2 байта)	Флаги(*) блока
HEAD_SIZE	2 байта	Размер блока
ADD_SIZE	4 байта	Добавление к размеру блока

Во всех блоках следующие биты в HEAD_FLAGS (*) имеют одинаковое значение:

– предпоследний 14-й (смещение 0x4000) — если = true (*), то старые версии RAR будут игнорировать этот блок и удалять его при изменении архива; иначе блок копируется в новый архивный файл при изменении архива;

– последний 15-й (смещение 0x8000) — если = true, то поле ADD_SIZE присутствует в блоке, иначе — отсутствует.

Заявленные типы блоков (возможные значения HEAD_TYPE):

– 0x72 (*) — блок-маркер;

– 0x73 — заголовок архива;

– 0x74 — заголовок файла;

– 0x75 — заголовок комментария старого типа;

– 0x76 — электронная подпись старого типа;

– 0x77 — субблок старого типа;

– 0x78 — информация для восстановления старого типа;

– 0x79 — электронная подпись старого типа;

– 0x7A — субблок.

Форматы блоков

Блок-маркер (MARK_HEAD)

– HEAD_CRC = 0x6152;

– HEAD_TYPE = 0x72;

– HEAD_FLAGS = 0x1a21;

– HEAD_SIZE = 0x0007;

Заголовок архива (MAIN_HEAD)

– HEAD_CRC = CRC полей от HEAD_TYPE до RESERVED2;

– HEAD_TYPE = 0x73;

– HEAD_FLAGS (=16 бит):

0-й бит (*) (смещение 0x0001 (*)) — Атрибут тома (том многотомного архива) (*)

1-й бит (смещение 0x0002) — Присутствует архивный комментарий (RAR 3.x использует отдельный блок комментария и не устанавливает этот флаг)

2-й бит (смещение 0x0004) — если = true, то архив заблокирован для изменений

3-й бит (смещение 0x0008) — если = true, то это — непрерывный (solid) архив

4-й бит (смещение 0x0010) — если = true, то используется новая схема именования томов ('volname.partN.rar'), иначе — старая ('volname.rN')

5-й бит (смещение 0x0020) — Присутствует информация об авторе или электронная подпись (AV) (RAR 3.x не устанавливает этот флаг)

6-й бит (смещение 0x0040) — Присутствует информация для восстановления

7-й бит (смещение 0x0080) — Заголовки блоков зашифрованы

8-й бит (смещение 0x0100) — Первый том (устанавливает только RAR 3.0 и выше)

Остальные биты (с 9 по 15-й) зарезервированы для внутреннего использования

– HEAD_SIZE (*) = Общий размер архивного заголовка, включая архивные комментарии;
– RESERVED1 (2 байта) - Зарезервировано;
– RESERVED2 (4 байта) - Тоже зарезервировано;

9.2 Структура ZIP архива

ZIP файл состоит из трех областей:

– сжатые/несжатые данные, (последовательность структур Local File Header, сами данные и необязательных Data descriptor);
– центральный каталог (последовательность структур Central directory file header);
– описание центрального каталога (End of central directory record).

С начала файла идет набор из Local File Header, непосредственно данные и (необязательно) структура Data descriptor. Затем структуры типа Central directory file header для каждого файла и папки в ZIP архиве и завершает все это структура End of central directory record.

Local File Header – используется для описания метаданных файла (имя файла, контрольная сумма, время и дата модификации, сжатый/несжатый размер). Как правило сразу после этой структуры следует содержимое файла.

struct LocalFileHeader

– uint32_t signature; // Обязательная сигнатура, равна 0x04034b50
– uint16_t versionToExtract; // Минимальная версия для распаковки
– uint16_t generalPurposeBitFlag; // Битовый флаг
– uint16_t compressionMethod; // Метод сжатия (0 - без сжатия, 8 - deflate)
– uint16_t modificationTime; // Время модификации файла
– uint16_t modificationDate; // Дата модификации файла
– uint32_t crc32; // Контрольная сумма
– uint32_t compressedSize; // Сжатый размер
– uint32_t uncompressedSize; // Несжатый размер
– uint16_t filenameLength; // Длина название файла
– uint16_t extraFieldLength; // Длина поля с дополнительными данными

– uint8_t *filename; // Название файла (размером filenameLength)

– uint8_t *extraField; // Дополнительные данные (размером extraFieldLength)

Сразу после этой структуры идут данные размером compressedSize при использовании сжатия или размером uncompressedSize в противном случае. Иногда бывает невозможно вычислить данные на момент записи LocalFileHeader, тогда в crc32, compressedSize и uncompressedSize записываются нули третий бит в generalPurposeBitFlag ставится в единицу и после LocalFileHeader добавляется структура типа Data descriptor.

Data descriptor

Если по какой-то причине содержимое файла невозможно создать одновременно с заголовком типа Local File Header, то сразу после него следует структура Data descriptor, где идет находится дополнение метаданных для Local File Header (контрольная сумма, сжатый/несжатый размер). Откровенно говоря, мне такие файлы не попадались, поэтому больше того, чем написано в википедии сказать не могу.

struct DataDescriptor

– uint32_t signature; // Необязательная сигнатура, равна 0x08074b50

– uint32_t crc32; // Контрольная сумма

– uint32_t compressedSize; // Сжатый размер

– uint32_t uncompressedSize; // Несжатый размер

Central directory file header

Расширенное описание метаданных файла. Содержит дополненную версию LocalFileHeader (добавляются поля номер диска, файловые атрибуты, смещение до Local File Header от начала ZIP файла).

struct CentralDirectoryFileHeader

// Обязательная сигнатура, равна 0x02014b50

– uint32_t signature; // Версия для создания

– uint16_t versionMadeBy; // Минимальная версия для распаковки

– uint16_t versionToExtract; // Битовый флаг

– uint16_t generalPurposeBitFlag; // Метод сжатия (0 - без сжатия, 8 - deflate)

– uint16_t compressionMethod; // Время модификации файла

– uint16_t modificationTime; // Дата модификации файла

– uint16_t modificationDate; // Контрольная сумма

– uint32_t crc32; // Сжатый размер

– uint32_t compressedSize; // Несжатый размер

– uint32_t uncompressedSize; // Длина название файла

– uint16_t filenameLength; // Длина поля с дополнительными данными

– uint16_t extraFieldLength; // Длина комментариев к файлу

– uint16_t fileCommentLength; // Номер диска

– uint16_t diskNumber; // Внутренние аттрибуты файла

– uint16_t internalFileAttributes; // Внешние аттрибуты файла

– uint32_t externalFileAttributes; // Смещение до структуры LocalFileHeader

– uint32_t localFileHeaderOffset; // Имя файла (длиной filenameLength)

– uint8_t *filename; // Дополнительные данные (длиной extraFieldLength)

– uint8_t *extraField; // Комментарий к файла (длиной fileCommentLength)

– uint8_t *fileComment;

End of central directory record (EOCD)

Эта структура записывается в конце файла. Содержит следующие поля: номер текущего диска, количество записей Central directory file header в текущем диске, общее количество записей Central directory file header.

// Обязательная сигнатура, равна 0x06054b50

– uint32_t signature; // Номер диска

– uint16_t diskNumber; // Номер диска, где находится начало Central Directory

– uint16_t startDiskNumber; // Количество записей в Central Directory в текущем диске

– uint16_t numberCentralDirectoryRecord; // Всего записей в Central Directory

– uint16_t totalCentralDirectoryRecord; // Размер Central Directory

– uint32_t sizeOfCentralDirectory; // Смещение Central Directory

– uint32_t centralDirectoryOffset; // Длина комментария

– uint16_t commentLength; // Комментарий (длиной commentLength)

– uint8_t *comment;

Папки в ZIP файле представлены двумя структурами Local File Header и Central directory file header с нулевым размером и контрольной суммой. Название папки заканчивается слешем «/».

9.3 Рефакторинг старого кода :: COEX

Рефакторинг — это процесс улучшения написанного ранее кода путем такого изменения его внутренней структуры, которое не влияет на внешнее поведение.

Во многом при рефакторинге лучше полагаться на интуицию, основанную на опыте. Тем не менее имеются некоторые видимые проблемы в коде, требующие рефакторинга:

– дублирование кода;

– длинный метод;

– длинный список параметров;

– «завистливые» функции — это метод, который чрезмерно обращается к данным другого объекта;

– избыточные временные переменные;

– классы данных;

– несгруппированные данные.

9.3.1 Рефакторинг плагина Pidgin

– Удаление дублирующегося кода

– Вынесение блоков кода отвечающих за разбор xml, html логов из TaskPidginWin::execute в отдельные функции (processingLogPidgin, processingContactPidgin, processingAccountPidgin).

– Замена алгоритма поиска файлов, на более понятный

– Использование qDebug вместо std::cout

9.3.2 Рефакторинг плагина Skype

– Удаление цикличного подключения к БД.
– Замена алгоритма поиска файлов, на более понятный.
– Использование qDebug вместо std::cout.

9.4 Совместимость плагинов под формат NoSql БД Solr

Для совместимости добавления в БД solr необходимо выполнить требования, указанные в источнике [**?**].

Документ добавляющий запись в БД имеет следющий формат:

Корневой тег add, затем каждая запись/событие помещается в тег doc, где далее/глубже лежат field с нашими данными. Атрибуты field name определяются разработчиком, в зависимости от необходимости, однако каждая запись должна иметь уникальный id. Так же добавляется в sceme.xml на сервере solr, для того чтобы БД знала какие данные в нее импортируются, и как с ними работать.

Старый формат

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Messages  Messenger=\"pidgin\">
    <info_account name=\"\" email=\"fox.user.3@gmail.com/\" password
        =\"kpdroscfozyyvsyk\" protocol=\"prpl-jabber\"/>
    <info_account name=\"Igor Polyakov\" email=\"fox.user.3@gmail.com\"
        password=\"this\_is\_real\_passowrd\" protocol=\"prpl-vkcom\"/>
    <info_account name=\"Igor Polyakov\" email=\"fox.user.3@gmail.com\"
        password=\"this\_is\_real\_passowrd\" protocol=\"prpl-vkcom\"/>
</Messages >
```

Новый формат

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<add>
    <doc>
        <field name=\"id\">pidgin\_24d7a3ebd9f601666a7ba27225e71854
            </field>
        <field name=\"doc_type\">account</field>
        <field name=\"application\">pidgin</field>
        <field name=\"account_id\"></field>
        <field name=\"account_mail\">fox.user.3@gmail.com/</field>
        <field name=\"account_protocol\">prpl-jabber</field>
        <field name=\"account_password\">kpdroscfozyyvsyk</field>
    </doc>
</add>
```

10 Механизм взаимодействия тасков с основной программой

После увеличения количества тасков и спектра решамых ими задач было решено переделать механизм взаимодействия тасков с основной частью программы. Так как для некоторых тасков необ-

ходимо передавать дополнительные входные данные, было принято решение реализации более гибкого решения. Каждый таск возвращает экземпляр класса, который реализует все необходимые функции. Такой подход позволяет так же реализовать систему настройки работы при помощи входных параметров, от которой пришлось отказаться после перехода на плагиню структуру.

Данный подход был реализован и на данный момент все таски работают по описанному выше принципу. Система разбора флагов на данный момент находиться в стадии разработки. На данный момент реализованы флаги '-i' и '-o' для указания входной и выходной дирректорий, а так же флаг без аргментов '-d' для включения отладочного режима. Но данный режим поддерживают не все таски, так как не проведен полный рефакторинг кода.

Так же ввиду такого подхода пришлось усложинить механизм работы с записью config, которая выродилась в отдельный объект и передаеться в конструктор класса-таска.

11 Механизм разбора флагов

Идея работы данного механизма не изменилась, мы перечисляем параметры для основного приложения, после чего указываем имя таска оканчивающиеся символом ':', после чего передаем флаги для данного таска. Внутри программы же мы разбиваем строку на "основные флаги" которая разбирается сразу же. остальная часть разбиваеться на строки вида "<taskname>: <flags and values>", которая помещаеться в QMap<QString, QString>, для дальнейшей передачи в таск как часть конфига.

Разбора параметров производиться при помощи команды c++ getargs, которая сама разбирает заданную строку, основываясь на информации о том, какие флаги должны присутствовать в строке и есть ли у них аргументы.

12 Config класс

Структура config, содержащая три поля: inputFolder, outputFolder, OS, превратилась в отдельный класс, ввиду увиличения количества полей (например добавления QMap с описанием всех аргументов тасков и значение флага debug), так же данный класс сожержит методы по разбору входных параметров и геттеры, для доступа к значению полей класса.

КО_КО_КО хз чо писать

13 TaskGetDictionary

Часто бывает так, что пользователи создают на своих персонаьлных компьютерах разные файлы, в которые записывают различные пароли. Появилась идея создания словаря паролей, на основании символьных последовательностей, найденных в файлах на образе. В данный момент реализован таск, который "читает" все текстовые файлы и создает на их основе словарь с последовательностями символов, который записываеться в SQLite базу данных. В дальнейшем планируеться расширить перечень файлов.

Алгоритм работы данного таска представлен на рисунке 13.1

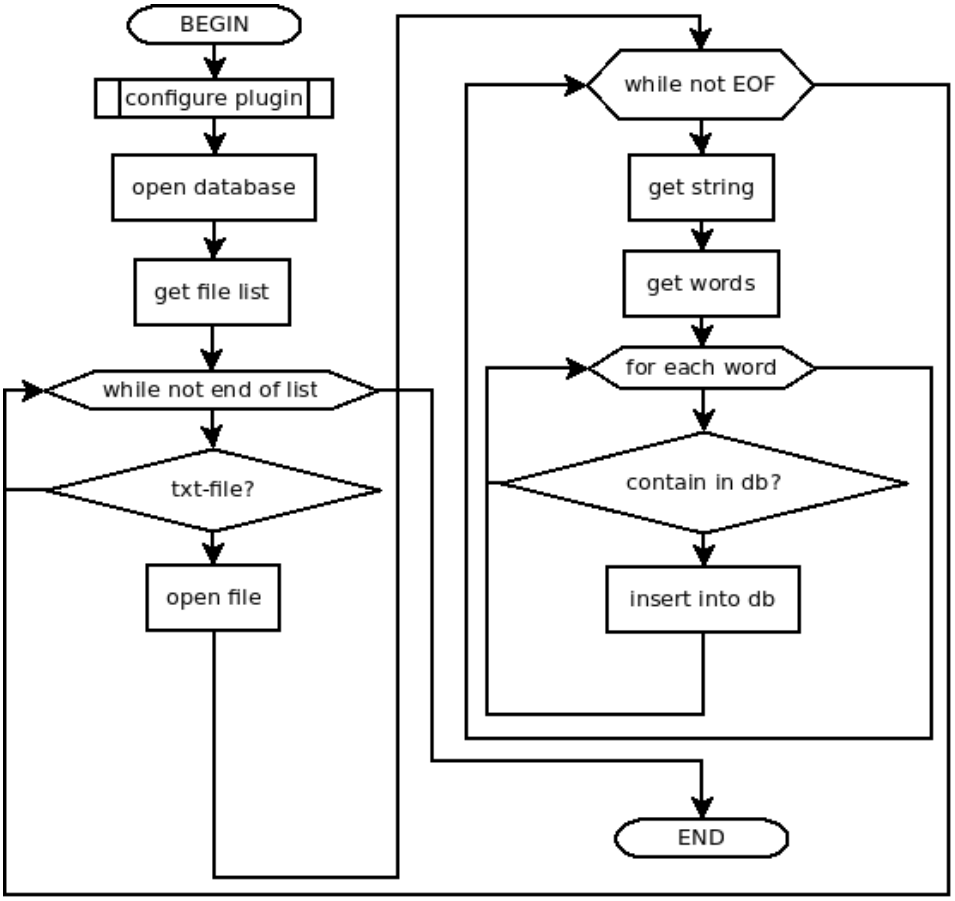

Рисунок 13.1 – алгоритм работы таска getDictionary

Заключение

В данном семестре нашей группой была выполнена часть работы по созданию автоматизированного программного комплекса для проведения компьютерной экспертизы, проанализированы дальнейшие перспективы и поставлены цели для будущих поколений.

Список использованных источников

1 Федотов Николай Николаевич. Форензика - компьютерная криминалистика. Юрид. мир, 2007. 432 с.

2 Scott Chacon. Pro Git : professional version control. 2011. URL: http://progit.org/ebook/progit.pdf.

3 С.М. Львовский. Набор и вёрстка в системе LaTeX. МЦНМО, 2006. С. 448.

4 И. А. Чеботаев, П. З. Котельников. LaTeX 2_ε по-русски. Сибирский Хронограф, 2004. 489 с.

5 Qt Documentation [Электронный ресурс] // qt-project.org:[сайт]. 2013. URL: http://qt-project.org/doc.

6 Всё о кроссплатформенном программировании - Qt [Электронный ресурс] // doc.crossplatform.ru:[сайт]. 2013. URL: http://doc.crossplatform.ru/qt.

7 Справочник по XML-стандартам [Электронный ресурс] // msdn.microsoft.com:[сайт]. URL: http://msdn.microsoft.com/ru-ru/library/ms256177(v=vs.110).aspx.

8 Windows Event Log Format [Электронный ресурс] // whitehats.ca:[сайт]. 2005. URL: http://www.whitehats.ca/main/members/Malik/malik_eventlogs/malik_eventlogs.html.

9 Официальное сообщество Skype [Электронный ресурс] // Community Skype:[сайт]. URL: http://community.skype.com.

10 Михайлов Игорь Юрьевич. Компьютерно-техническая экспертиза [Электронный ресурс] // Компьютерно-техническая экспертиза: [сайт]. [2013]. URL: http://computer-forensics-lab.org.

11 Официальное сообщество Pidgin // developer.pidgin.im:[сайт]. URL: https://developer.pidgin.im.

12 Алексей Николаевич Валиков. Технология XSLT. БХВ-Петербург, 2002. 544 с.

13 Сертификация программного обеспечения, центр сертификации Мостест [Электронный ресурс] // mostest.su:[сайт]. URL: http://mostest.su/uslugi-po-sertifikacii-oformlenie-sertifikatov-otkaznyx-pisem-i-dr/sertifikaciya-programmnogo-obespecheniya.

14 Сертификация средств защиты информации (СЗИ) по требованиям безопасности информации: сертификация и сертификационные испытания в системах сертификации ФСТЭК России, Минобороны России, ФСБ России, «АйТи-Сертифика» [Электронный ресурс] // Эшелон:[сайт]. URL: http://www.npo-echelon.ru/services/certification.

Приложение А
(Обязательное)
Компакт-диск

Компакт-диск содержит:

– электронную версию пояснительной записки в форматах *.tex и *.pdf;
– актуальную версию программного комплекса для проведения компьютерной экспертизы;
– тестовые данные для работы с программным комплексом.