МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Базовая кафедра №252 – информационной безопасности

ЛАБОРАТОРНАЯ РАБОТА №2 ПО ПРЕДМЕТУ «РАЗРУШАЮЩИЕ ПРОГРАММНЫЕ ВОЗДЕЙСТВИЯ»

Студент группы ККСО-01-20 *Семин В.В.*

Преподаватель *Старший преподаватель*
Трошков Вадим Евгеньевич

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ..... 3

1 ОПИСАНИЕ ФУНКЦИЙ, РЕАЛИЗУЮЩИХ ЗАЯВЛЕННЫЙ ФУНКЦИОНАЛ..... 4

2 ПЕРЕЧЕНЬ НЕИСПОЛЬЗУЕМЫХ ФУНКЦИЙ, НЕ СООТВЕТСТВУЮЩИХ ДОКУМЕНТАЦИИ ФУНКЦИЙ, ФУНКЦИЙ, СОДЕРЖАЩИХ ПОДОЗРИТЕЛЬНЫЙ КОД..... 10

3 ПЕРЕЧЕНЬ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ И ФУНКЦИИ, В КОТОРЫХ ОНИ ИСПОЛЬЗУЮТСЯ..... 19

4 СПИСОК ИЗБЫТОЧНЫХ ФАЙЛОВ И ДИРЕКТОРИЙ..... 30

5 БЛОК-СХЕМА ПРОГРАММЫ..... 31

6 СПИСОК ИНСТРУМЕНТОВ, ИСПОЛЬЗУЕМЫХ ПРИ АНАЛИЗЕ…32

ЗАКЛЮЧЕНИЕ..... 33

ВВЕДЕНИЕ

Поиск недекларированных возможностей в программе. Проводится на основе руководящего документа (РД НДВ-99).

Считаем, что документация полная.

Необходимо:

1) определить избыточные функции и файлы;
2) построить дерево вызовов программы;
3) определить все глобальные переменные и установить, в каких фрагментах программы они используются;
4) определить функции, которые не соответствуют документации или выполняют подозрительные действия и провести их анализ.

1 ОПИСАНИЕ ФУНКЦИЙ, РЕАЛИЗУЮЩИХ ЗАЯВЛЕННЫЙ ФУНКЦИОНАЛ

Файл Form1.cs описывает класс *Form1*, наследуемый от класса *Form*, который является основной формой приложения Windows Forms.

Конструктор класса Form1 (файл Form1.cs строка 32) выполняет несколько ключевых задач при создании экземпляра формы.

1) Инициализация компонентов. Вызов *InitializeComponent* инициализирует все компоненты формы, такие как кнопки, текстовые поля и дерево (TreeView).
2) Установка заголовка формы на строку *"Коллекция терминов"*
3) Инициализация сессии Nhibernate. Конструктор принимает ссылку на *ISessionFactory* и сохраняет её в поле *_session*. Это позволяет форме взаимодействовать с базой данных через Nhibernate.
4) Установка начальных значений. Устанавливаются начальные значения для полей *selectedIndex*, *selectedParent*, *selectedSource*, и *selectedCategory*. Эти поля используются для отслеживания текущего выбранного элемента в TreeView.
5) Загрузка данных. Вызывается метод *GetTermList*, который заполняет TreeView данными из базы данных. Это позволяет пользователю сразу увидеть структуру данных при открытии формы.
6) Назначение обработчиков событий. Назначаются обработчики событий для элементов управления. *textBox1_TextChanged* обрабатывает изменение текста в *textBox*. *TreeView1_AfterSelect* обрабатывает выбор узла в *TreeView*.

Метод *RefreshTextBox* (файл Form1.cs строка 50) очищает текстовое содержимое двух элементов управления *RichTextBox*, которые, используются для отображения информации на форме.

Метод *TreeView1_AfterSelect* (файл Form1.cs строка 203) отображает на экране определение и источник запрошенного пользователем термина.

Метод *textBox1_TextChanged* (файл Form1.cs строка 256) осуществляет поиск термина по мере набора пользователем текста в окне поиска.

Метод *GetTermList* (файл Form1.cs строка 279) в этом коде отвечает за загрузку и отображение иерархической структуры терминов из базы данных в элемент управления TreeView.

Метод *button4_Click* (файл Form1.cs строка 463) в *Form1* отвечает за восстановление базы данных из резервной копии.

Метод *button5_Click* (файл Form1.cs строка 477) в *Form1* отвечает за экспорт всей базы терминов в PDF-файл.

Метод *button8_Click* (файл Form1.cs строка 550) в *Form1* отвечает за навигацию по результатам поиска.

Метод *checkBox1_CheckedChanged* (файл Form1.cs строка 561) отвечает за переключение между отображением только действующих источников и всех источников.

Класс *MeaningMap* (файл MeaningMap.cs, строка 6) описывает, как сущность *Meaning* (определение термина) должна отображаться в базе данных.

Конструктор *MeaningMap* (файл MeaningMap.cs, строка 8):

1) Определяет первичный ключ для таблицы, сопоставляя поле *Id* в классе *Meaning*.
2) Сопоставляет свойство *Text* с колонкой в базе данных, хранящей текст значения термина.
3) В таблице *Meaning* создаётся внешний ключ *TermId*, указывающий на одну запись из таблицы *Term*.
4) В таблице *Meaning* создаётся внешний ключ *SourceId*, указывающий на одну запись из таблицы *Source*.

5) Сопоставляет поле *Location*, хранящее дополнительную информацию о местоположении значения в источнике.

Класс *SourceMap* (файл SourceMap.cs, строка 6) описывает, как сущность *Source* (источник) должна отображаться в базе данных.

Конструктор SourceMap (файл SourceMap.cs, строка 8):

1) Определяет первичный ключ для таблицы, сопоставляя поле *Id* в классе *Source*.
2) Сопоставляет свойство *Name* с колонкой в базе данных, хранящей название источника.
3) Сопоставляет свойство *Active* с колонкой, указывающей на актуальность источника.
4) Создаёт связь многий-ко-многим между *Source* и *Meaning*. Это может быть избыточным, так как по логике приложения определение ссылается только на один источник (см. описание *MeaningMap*). Тут скорее нужна связь многий-ко-одному.
5) Добавляет внешний ключ *CategoryId* в таблицу *Source*, связывая её с *Category.*

Класс *TermMap* (файл TermMap.cs, строка 6) описывает, как сущность *Term* (термин) должна отображаться в базе данных.

Конструктор *TermMap* (файл TermMap.cs, строка 8)

1) Определяет первичный ключ для таблицы, сопоставляя поле *Id* в классе *Term*.
2) Сопоставляет свойство *Name* с колонкой в базе данных, хранящей название термина.
3) сопоставляет свойство *EngName* с колонкой, хранящей название термина на английском языке.
4) Сопоставляет свойство *Abbrev* с колонкой, хранящей аббревиатуру термина.

5) Сопоставляет свойство *Active* с колонкой, указывающей на актуальность термина.
6) Создаёт связь многий-ко-многим между *Term* и *Meaning.* Это может быть избыточным, так как по логике приложения определение ссылается только на один термин (см. описание *MeaningMap*). Тут скорее нужна связь многий-ко-одному.

Класс *Meaning* (файл Meaning.cs строка 4) описывает определение термина.

Метод *JoinTerm* (файл Meaning.cs строка 12) добавляет определение к термину, если его нет.

Метод *JoinSource* (файл Meaning.cs строка 20) определение к источнику, если его нет.

Метод *BondTermSource* (файл Meaning.cs строка 29) выполняет *JoinTerm* и *JoinSource*.

Метод *ToString* (файл Meaning.cs строка 42) возвращает название категории.

Класс *Source* (файл Source.cs строка 4) описывает источники.

Конструктор класса *Source* (файл Source.cs строка 12) инициализирует список определений.

Метод *AddTerm* (файл Source.cs строка 17) создаёт новое определение и привязывает его к термину и источнику.

Метод *TurnOff* (файл Source.cs строка 26) помечает источник и все его определения как неактуальные. Если термин, к которому привязано определение из источника имеет только одно определение, то он также помечается как неактуальный.

Метод *TurnOn* (файл Source.cs строка 50) помечает источник как актуальный.

Класс *Term* (файл Term.cs строка 4) описывает источники.

Конструктор класса *Term* (файл Term.cs строка 6) инициализирует список определений.

AddSource (файл Term.cs строка 20). Добавление нового определения.

TurnOff (файл Term.cs строка 29). Пометка термина неактуальным.

TurnOn (файл Term.cs строка 34). Пометка термина актуальным.

Класс *Program* (файл Program.cs строка 15) является набором функций, которые непосредственно являются пользовательским интерфейсом.

Функция *Main* (файл Program.cs строка 21) является точкой входа в программу. Выполняет следующие действия:

1) Создаёт фабрику сессий Nhibernate.
2) Запускает WinForms-приложение.
3) Передаёт *sessionFactory* в форму *Form1* (главное окно приложения).
4) После закрытия формы удаляет резервную копию БД.

Функция *ResetDatabase* (файл Program.cs строка 84) восстанавливает состояние БД из резервной копии и перезагружает БД.

Функция *CreateTerm* (файл Program.cs строка 96) создаёт новый объект термина.

Функция *CreateSource* (файл Program.cs строка 107) создаёт новый объект источника.

Функция *CreateSessionFactory* (файл Program.cs строка 122) отвечает за настройку и создание фабрики сессий NHibernate для работы с SQLite-базой данных:

1) Создаёт резервную копию БД.
2) Настраивает NHibernate через Fluent API.
3) Создаёт/обновляет схему БД.

Функция *BuildSchema* (файл Program.cs строка 134) отвечает за автоматическое создание или обновление структуры базы данных (таблиц, индексов, связей) на основе маппингов Nhibernate:

1) Проверяет существование файла БД.
2) Генерирует и выполняет DDL-скрипты.

Функция *CreateMeaningAndSource* (файл Program.cs строка 158) отвечает за создание и связывание сущностей *Term*, *Meaning*, *Source* и *Category* на основе данных из *JSON*.

Три функции *FindRepetitions* (файл Program.cs строки 194, 206, 218) выполняют поиск, соответственно, повторяющихся источников, категорий, терминов.

2 ПЕРЕЧЕНЬ НЕИСПОЛЬЗУЕМЫХ ФУНКЦИЙ, НЕ СООТВЕТСТВУЮЩИХ ДОКУМЕНТАЦИИ ФУНКЦИЙ, ФУНКЦИЙ, СОДЕРЖАЩИХ ПОДОЗРИТЕЛЬНЫЙ КОД

Методы *AESEncryptDLL* и *AESDecryptDLL* класса *Form1* (файл Form1.cs, строки 56 и 84) выполняют шифрование массива байт с помощью AES-128-CBC. Их графы вызовов, , полученные с помощью утилиты Understand, представлены на Рисунках 2.1 и 2.2.

Рисунок 2.1 — Граф вызовов функции AESEncryptDLL.

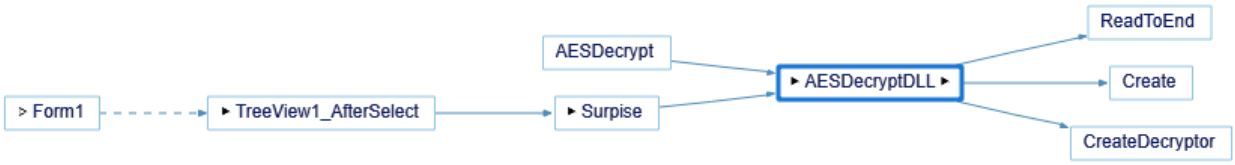

Рисунок 2.2 — Граф вызовов функции AESDecryptDLL.

Метод *Surpise* в классе *Form1* (файл Form1.cs, строка 113) выполняет несколько действий, связанных с дешифрованием и выполнением встроенного ресурса DLL.

1) Метод использует *ResourceManager* для извлечения зашифрованного ресурса surprise из сборки. Этот ресурс представляет собой массив байтов, который содержит зашифрованную DLL.
2) Извлечённый массив байтов encdll содержит зашифрованные данные. Первые 213 байта удаляются.
3) Оставшиеся данные дешифруются с использованием метода *AESDecryptDLL*. Для дешифрования используется заранее определённый

ключ (ac1e6fb65060175cc7f60e32bead9701177ff8808ecd77f19a3cc889867a82f0) и вектор инициализации (00000000000000000000000000000000).

4) Дешифрованные данные затем преобразуются из строки Base64 обратно в массив байтов, представляющий собой DLL.
5) Дешифрованная DLL сохраняется на диск под именем surprise.dll.
6) Используется *Process* для выполнения дешифрованной DLL с помощью rundll32.exe. В зависимости от разрядности операционной системы (64-битная или 32-битная) выбирается соответствующий путь к rundll32.exe.
7) В качестве параметра командной строки передаётся значение, основанное на *selectedIndex*. Это значение используется для выбора варианта выполнения DLL.
8) После выполнения DLL временный файл surprise.dll удаляется с диска.

Данный метод не соответствует описанному в документации функционалу программы. Он является программной закладкой. Выполнение дешифрованного кода может быть рискованным, особенно если источник кода не проверен. Это может привести к выполнению вредоносного кода.

Граф вызовов функции *Surpise*, полученный с помощью утилиты Understand. представлен на Рисунке 2.1.

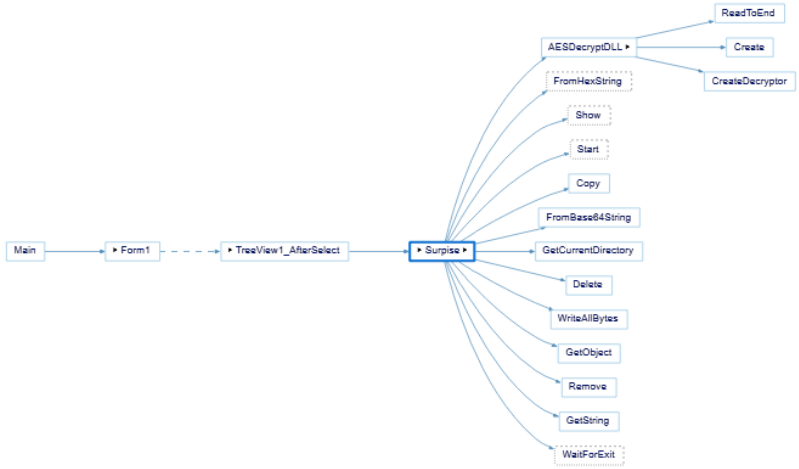

Рисунок 2.3 — Граф вызовов функции Surpise.

Метод AESDecrypt (файл Form1.cs, строка 178)

1) Читает файл RevDll.dll.enc.
2) Преобразует его в строку (что странно, так как DLL — это бинарный файл).
3) Шифрует эту строку с помощью *AESEncryptDLL*. Используются те же ключ и вектор инициализации, что и методе *Surpise.*
4) Создаёт файл surprise.resources с зашифрованными данными.
5) Пытается расшифровать данные и сравнить с исходными.
6) Если исходные данные (*plaintext*) совпадают с расшифрованными (*decrypted*), метод AESDecrypt выполняет следующие действия: выводит сообщение *"true\n"* в *richTextBox1*. Дополнительно выводит размер и информацию о зашифрованных данных.

Данный метод не используется в коде. Однако он выполняет подозрительные действия. Возможно злоумышленник предполагает некий альтернативный способ его вызова (через другие уязвимости в системе).

Метод *TreeView1_AfterSelect*, как и было сказано в 1 главе, при нормальной работе отображает на экране определение и источник запрошенного пользователем термина. Но в случае возникновения исключения при пирсинге значения *selectedParent* активизирует заложенную в код программную закладку (вышеописанный метод *Surpise*).

Следовательно, для активизации программной закладки нужно выбрать узел дерева, который не имеет родителя, то есть либо категорию, либо термин.

Метод *button1_Click* (файл Form1.cs, строка 352) является обработчиком нажатия кнопки. Он вызывает *Form2*, которая реализует функционал изменения значения термина. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных.

Метод *button2_Click* (файл Form1.cs, строка 373) в *Form1* отвечает за удаление выбранного элемента (категории, источника или термина) из базы данных и дерева *TreeView*. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных.

Метод *button3_Click* (файл Form1.cs, строка 442) в *Form1* отвечает за добавление нового источника (*Source*) в выбранную категорию или создание новой категории (*Category*), если ничего не выбрано. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных. Кроме этого документация не предполагает работу программы с такими категориями источников как ГОСТ. ФЗ и так далее, а только деление их на действующие и не действующие.

Метод *button6_Click* (файл Form1.cs, строка 524) в *Form1* отвечает за открытие формы для добавления нового термина в выбранный источник. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных.

Метод *button7_Click* (файл Form1.cs, строка 537) в *Form1* отвечает за открытие формы для добавления нового источника или категории. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных. Кроме этого документация не предполагает работу программы с такими категориями источников как ГОСТ. ФЗ и так далее, а только деление их на действующие и не действующие.

Класс *Form2* (файл Form2.cs, строка 15) реализует функционал редактирования определения термина. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных.

Конструктор класса *Form2* (файл Form2.cs, строка 21):

1) Устанавливает заголовок формы ("Изменение термина").
2) Добавляет подпись "Статья\nиз источника".
3) Сохраняет ссылки на *Form1, tree* и *node.*
4) Переносит данные из *Form1*: название термина, определение, источник.

button1_Click (файл Form2.cs, строка 51). Обработчик нажатия кнопки. Сохраняет изменения термина в базу данных.

Класс *Form3* (файл Form3.cs, строка 15) реализует функционал добавления нового термина. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных.

Конструктор класса Form3 (файл Form3.cs, строка 21):

1) Сохраняет ссылку на переданный объект *Form1* в поле *parent*.
2) Сохраняет ссылку на переданный *TreeView*.
3) Создаёт и настраивает элементы интерфейса.
4) Очищают *comboBox1* (источники) и *comboBox2* (категории).
5) Перебирает все узлы верхнего уровня (категории) в *TreeView*. Добавляет имена категорий в *comboBox2*.
6) Устанавливает текстовое значение для *comboBox1* на *"Сначала выберите категорию"*, чтобы показать пользователю инструкцию.
7) Изменяет текст *label6* на *"Статья\nиз источника"*.
8) Подписывает метод *comboBox1_SelectedIndexChanged* на событие выбора (*SelectedIndexChanged*).

9) Подписывает метод *comboBox2_SelectedIndexChanged* на событие выбора (*SelectedIndexChanged*).

Метод *comboBox1_SelectedIndexChanged* (файл Form3.cs, строка 41) вызывается каждый раз, когда пользователь выбирает новый элемент в *comboBox1*. Получает выбранное пользователем значение из *comboBox1*. Сохраняет его в переменную *selectedState*.

Метод *comboBox2_SelectedIndexChanged* (файл Form3.cs, строка 47) срабатывает каждый раз, когда пользователь выбирает новую категорию в *comboBox2* (выпадающем списке категорий). Его задача — обновить *comboBox1* (список источников), чтобы отобразить только источники, принадлежащие выбранной категории.

Метод *button1_Click* (файл Form3.cs, строка 58) срабатывает при нажатии кнопки. Его основная задача — добавить новый термин в базу данных и обновить *TreeView*.

Класс *Form4* (файл Form4.cs, строка 14) реализует функционал редактирования источников и категорий. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных. Кроме этого документация в целом предполагает только две категории: действующие и недействующие.

Конструктор класса *Form4* (файл Form4.cs, строка 20):

1) Инициализирует поля.
2) Определяет режим работы. В зависимости от значений *selectedIndex* и *selectedParent* в *Form1*, форма работает в одном из двух режимов: редактирование источника или редактирование категории.
3) Заполняет *richTextBox1* и *richTextBox2* данными.

Метод *button1_Click* (файл Form4.cs, строка 78) выполняет сохранение изменений в *TreeView* и базе данных с помощью Nhibernate.

Класс Form5 (файл Form5.cs, строка 14) реализует функционал добавления источников и категорий. Это не соответствует заявленному в документации функционалу программы. В документации сказано, что пользователь не может изменять текущую базу данных. Кроме этого документация в целом предполагает только две категории: действующие и недействующие.

Конструктор класса *Form5* (файл Form5.cs, строка 20):

1) Сохраняет ссылку на Form1, чтобы можно было работать с её данными.
2) Сохраняет ссылку на переданный TreeView, чтобы взаимодействовать с деревом категорий и источников.
3) Создаёт элементы интерфейса формы.
4) Заполняет *comboBox1* категориями.
5) Настраивает comboBox1. Привязывает обработчик события, который будет срабатывать при выборе категории.

Метод *button1_Click* (файл Form5.cs, строка 36) в классе *Form5* отвечает за добавление нового источника в выбранную категорию в дереве *TreeView* и сохранение данных в базу данных с использованием *Nhibernate*.

Метод *comboBox1_SelectedIndexChanged* в классе *Form5* срабатывает при изменении выбранного элемента в комбинированном списке *comboBox1*. Очищает текстовое поле в комбинированном списке. Получает строковое значение текущего выбранного элемента из комбобокса. По сути этот метод бесполезен, так как текст комбобокса автоматически обновится, а переменная, в которую сохраняется значение текущего выбранного элемента не используется. Следовательно, он избыточен.

Метод *button2_Click* в *Form5* отвечает за добавление новой категории в базу данных и в *TreeView*.

Класс *CategoryMap* (файл CategoryMap.cs строка 6) описывает, как сущность *Category* (категория источника) должна отображаться в базе данных. Не

соответствует документации, так как та предполагает деление источников только на действующие и не действующие.

Конструктор *CategoryMap* (файл CategoryMap.cs строка 8):

1) Указывает, что поле *Id* в классе *Category* является первичным ключом.
2) Определяет, что свойство *Name* будет отображаться как колонка в таблице БД.
3) Указывает, что свойство *Active* (актуальность) также будет храниться в БД.
4) Определяет один-ко-многим связь с коллекцией *Source* (источники).

Класс *Category* (файл Category.cs строка 3) описывает категорию источника (например ГОСТ или ФЗ). Не соответствует документации, так как по ней программа делит источники только на актуальные и неактуальные.

Конструктор *Category* (файл Category.cs строка 9) инициализирует список источников, относящихся к этой категории.

Метод *TurnOff* (файл Category.cs строка 14) помечает категорию и относящиеся к ней источники как неактуальные. Обновляет базу данных после этого.

Метод *TurnOn* (файл Category.cs строка 24) помечает категорию как актуальную. Обновляет базу данных после этого.

Метод *ChengeRelation* (файл Meaning.cs строка 35) отвязывает определение от источника и термина и привязывает его к новым источникам и терминам.

Метод *ChangeCategory* (файл Source.cs строка 40) отвязывает источник от категории и привязывает его к новой категории.

Функция *CreateCategory* (файл Program.cs строка 116) создаёт новый объект категории.

Функция *ReadJson* (файл Program.cs строка 148) отвечает за чтение и десериализацию JSON-файла (koval_terms.json) в объекты класса *Data*. Не соответствует документации, так как по ней программа не предусматривает входные данные. Кроме этого является избыточным, так как нигде не используется.

Классы *Data, DataMeaning, DataSource* (файл Program.cs строки 245, 254, 261), не смотря на яркие комментарии с обсценной лексикой, выполняет вполне понятную функцию. Они нужны для хранения полученных *ReadJson* данных. Не соответствуют документации, так как по ней программа не предусматривает входные данные. Кроме этого являются избыточным, так как полученные данные не используются.

3 ПЕРЕЧЕНЬ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ И ФУНКЦИИ, В КОТОРЫХ ОНИ ИСПОЛЬЗУЮТСЯ

Ниже описаны поля класса *From1*.

1) *queue* (файл Form1.cs строка 23). Хранит результаты поиска терминов. Использование:
   а) Конструктор *Form1*. Инициализация.
   б) *textBox1_TextChanged.* Очищается, затем заполняется при поиске.
   в) *button8_Click*. Навигация по результатам.
2) *selectedIndex* (файл Form1.cs строка 24). ID выбранного термина, источника или категории. Использование:
   а) Конструктор *Form1*. Устанавливается начальное значение -1.
   б) *Surpise*. Вывод сообщения *"You've choosen poorly! Your variant is {(selectedIndex % 4) + 1}"* после запуска вредоносной dll. Использование значения *(selectedIndex % 4) + 1* в качестве аргумента при запуске программной закладки.
   в) *TreeView1_AfterSelect*. Сохраняет ID. Проверяет значение нового ID. При корректном значении выводит термин на экран.
   г) *button1_Click*. Проверка того,сделал ли пользователь выбор.
   д) *button2_Click.* Проверка того,сделал ли пользователь выбор. Если да, то удаление термина по этому ID.
   е) *button3_Click*. Проверка того,сделал ли пользователь выбор.
   ж) *button1_Click* в *Form2*. Проверка того,сделал ли пользователь выбор.
   з) Конструктор *Form4*. Проверка того,сделал ли пользователь выбор.
   и) *button1_Click* в *Form4*. Проверка того,сделал ли пользователь выбор.
3) *selectedParent* (файл Form1.cs строка 25). ID источника (если выбран термин) или категории (если выбран источник), -1, если выбрана категория. Использование:
   а) Конструктор *Form1*. Устанавливается начальное значение -1.

б) *TreeView1_AfterSelect*. Сохраняет ID. Проверяет значение нового ID. При корректном значении выводит термин на экран. В случае ошибки парсинга нового значения устанавливает значение -1 и запускает *Surpise*.

в) *button1_Click*. Проверка того,сделал ли пользователь выбор.

г) *button2_Click*. В случае, если установлено корректное значение (не -1) достаёт название источника из БД.

д) *button3_Click*. Проверка того,сделал ли пользователь выбор.

е) *button2_Click* в *Form2*. В случае, если установлено корректное значение (не -1) достаёт название источника из БД.

ж)Конструктор *Form4*. Проверка того, сделал ли пользователь выбор.

з) *button1_Click* в *Form4*. Проверка того,сделал ли пользователь выбор.

4) *selectedSource* (файл Form1.cs строка 26). Использование:

а) Конструктор *Form1*. Устанавливается начальное значение — пустая строка.

Данная переменная нигде (кроме установки значения в конструкторе) не используется. Она является избыточной.

5) *SelectedCategory* (файл Form1.cs строка 27). ID категории источника, если выбран термин. В ином случае равен =1, так как в случае выбора источника ID категории хранится в *selectedParent*, в случае выбора категории в *selectedIndex*. Использование:

а) Конструктор *Form1*. Устанавливается начальное значение -1.

б) *TreeView1_AfterSelect*. Сохраняет ID. Проверяет значение нового ID. При корректном значении выводит термин на экран. В случае ошибки парсинга нового значения устанавливает значение -1.

в) *button1_Click*. Проверка того,сделал ли пользователь выбор.

г) *button2_Click*. В случае, если установлено корректное значение (не -1) достаёт категории источника из БД.

д) *button3_Click*. Проверка того,сделал ли пользователь выбор.

Не соответствует документации, так как по ней программа разделяет источники только на действующие и недействующие. Разделение на категории, такие как ГОСТ, ФЗ и так далее документация не предполагает.

6) *numberOfCat* (файл Form1.cs строка 28). Не используется. Является избыточной.
7) *SelectedSearch* (файл Form1.cs строка 29). Индекс текущего результата поиска. Используется:
   а) *textBox1_TextChanged*. Сброс при новом поиске.
   б) *button8_Click*. Навигация по результатам поиска.
8) *_session* (файл Form1.cs строка 31). Фабрика сессий Nhibernate. Используется:
   а) Конструктор *Form1*. Установка значения. Определяется аргументом конструктора *session*.
   б) *TreeView1_AfterSelect*. Открытие сессии для получения источника и определения термина.
   в) *GetTermList*. Открытие сессии для получения списка терминов.
   г) *button2_Click*. Открытие сессии для удаления элемента.
   д) *button4_Click*. Открытие сессии для сброса изменений БД.
   е) *button5_Click*. Открытие сессии для печати БД в PDF
   ж)*button1_Click* в *Form2*. Открытие сессии для изменения элемента.
   з) *button1_Click* в *Form3*. Открытие сессии для добавления элемента.
   и) *button1_Click* в *Form4*. Открытие сессии для изменения элемента.
   к) *button1_Click* в *Form5*. Открытие сессии для добавления элемента.
   л) *button2_Click* в *Form5*. Открытие сессии для добавления элемента.

Ниже описаны поля класса *From2.*

1) *parent* (файл Form2.cs строка 17). Хранит ссылку на родительскую форму *Form1*. Использование:

а) Конструктор *Form2.* Получает ссылку на *Form1* и сохраняет её в поле. Использует для доступа к элементам управления и полям родительской формы.

б) *button1_Click*. Используется для работы с NHibernate, получения *selectedIndex*, *selectedParent*, *selectedCategory* и взаимодействия с базой данных.

2) *node* (файл Form2.cs строка 18). Хранит выбранный узел *TreeView*, переданный из *Form1*. Использование:

а) Конструктор *Form2.* Устанавливается в зависимости от параметра *selectedNode.* Используется для установки текста в *richTextBox1*.

б) *button1_Click.* После сохранения изменений сбрасывается и вновь устанавливается как выбранный узел *TreeView*.

Вне конструктора не используется никак кроме обновления значения. Является избыточным.

3) *tree* (файл Form2.cs строка 19). Хранит ссылку на *TreeView*, переданный из *Form1*. Использование:

а) Конструктор *Form2.* Инициализируется значением из *Form1*.

б) *button1_Click*. Обновляется при изменении данных узла.

4) *indexNode* (файл Form2.cs строка 20). Хранит индекс элемента управления *treeView1* в *Form1.Controls*. Использование:

а) Конструктор *Form2*. Ищет *treeView1* в *Form1.Controls* и сохраняет его индекс.

В дальнейшем не используется, что делает это поле избыточным.

Ниже описаны поля класса *From3*.

1) *parent* (файл Form3.cs, строка 17). Хранит ссылку на родительскую форму *Form1*. Использование:

а) Конструктор *Form3*. Получает ссылку на Form1 и сохраняет её в поле.

б) *button1_Click*. Используется для работы с NHibernate, сохранения нового термина в базе и обновления списка терминов в *Form1*.

2) *node* (файл Form3.cs, строка 18). Является избыточным и может быть удалено.
3) *tree* (файл Form3.cs, строка 19). Хранит ссылку на *TreeView*, переданный из Form1. Использование:
   а) Конструктор *Form3*. Инициализируется значением из *Form1*.
   б) *comboBox2_SelectedIndexChanged*. Используется для отображения списка источников при выборе категории.
   в) *button1_Click*. Обновляется при добавлении нового термина, чтобы отразить изменения в *TreeView*.
4) *indexNode* (файл Form3.cs, строка 20). Не используется в коде, что делает это поле избыточным.

Ниже описаны поля класса *From4*.

1) parent (файл Form4.cs, строка 16). Хранит ссылку на родительскую форму *Form1*.
   а) Конструктор *Form4*. Получает ссылку на Form1 и сохраняет её в поле. Используется для доступа к данным, таким как *selectedIndex* и *selectedParent*.
   б) *button1_Click*. Используется для работы с NHibernate, получения данных о категории и источнике, а также для сохранения изменений.
2) *node* (файл Form4.cs, строка 17). Хранит выбранный узел *TreeView*, переданный из Form1. В конструкторе *Form4* инициализируется значением из *selectedNode*. Используется для установки текста в *richTextBox1* и *richTextBox2*. Вне конструктора поле *node* нигде не используется. Оно лишь принимает значение в конструкторе и затем не участвует в логике программы. Это делает его избыточным.

3) *tree* (файл Form4.cs, строка 18). Хранит ссылку на *TreeView*, переданный из *Form1*. Использование:
   а) Конструктор *Form4*. Инициализируется значением из *Form1*. Используется для работы с узлами дерева при редактировании категории или источника.
   б) *button1_Click*. Используется для обновления имени узла и его актуальности после сохранения изменений.
4) *indexNode* (файл Form4.cs, строка 19). Сохраняет индекс элемента управления *treeView1* в *Form1.Controls*. В конструкторе устанавливается его значение. В дальнейшем оно не используется. Избыточное.

Ниже описаны поля класса *From5*.

1) *parent* (файл Form5.cs, строка 16). Хранит ссылку на родительскую форму *Form1*. Использование:
   а) Конструктор *Form5*. Получает ссылку на *Form1* и сохраняет её в поле.
   б) button1_Click. Используется для работы с NHibernate и добавления нового источника.
2) *node* (файл Form5.cs, строка 17). Поле объявлено, но нигде в коде не используется.
3) *tree* (файл Form5.cs, строка 18). Хранит ссылку на *TreeView*, переданный из *Form1*. Использование:
   а) Конструктор *Form5*. Инициализируется значением из Form1.
   б) *button1_Click*. Используется для добавления нового источника в *TreeView*.
   в) *button2_Click*. Используется для добавления новой категории в *TreeView*.
   г) *comboBox1_SelectedIndexChanged*. Используется для выбора категории.

4) *indexNode* (файл Form5.cs, строка 19). Поле объявлено, но нигде в коде не используется.

Ниже описаны поля класса *Category*.

1) *Id* (файл Category.cs, строка 5). ID категории. Использование:
   а) *button2_Click* в *Form5*. Добавление новой категории.
2) *Name* (файл Category.cs, строка 6). Название категории. Использование:
   а) *button5_Click* в *Form1*. Печать в PDF.
   б) *button2_Click* в *Form5*. Добавление новой категории.
   в) *CreateCategory* в *Progam*. Добавление новой категории.
   г) *CreateMeaningAndSource* в *Progam*. Если объекта категории для добавляемого источника не существует, он создаётся.
   д) *FindRepetitions* в *Progam*. Поиск повторяющихся категорий.
3) *Source* (файл Category.cs, строка 7). Список источников из данной категории. Использование:
   а) Конструктор *Category*. Инициализация.
   б) *TurnOff*. Итерация по источникам для того, чтобы пометить их неактуальными.
   в) *button5_Click* в *Form1*. Печать в PDF.
   г) *CreateSource* в *Progam*. Добавление новой источника в категорию.
   д) *CreateMeaningAndSource* в *Progam*. Добавление новой источника в категорию.
   е) *ChangeCategory* в *Source*. Удаление источника из старой категории и добавление его в новую.
4) *Active* (файл Category.cs, строка 7). Актуальность/неактуальность категории. Использование:
   а) *TurnOff*. Категория помечается неактуальной.
   б) *TurnOn*. Категория помечается актуальной.
   в) *CreateCategory* в *Progam*. Добавление новой категории.

г) *CreateMeaningAndSource* в *Progam*. Если объекта категории для добавляемого источника не существует, он создаётся.

Ниже описаны поля класса *Meaning*.

1) *Id* (файл Meaning.cs, строка 6). ID определения. Явно в коде не используется, однако требуется для загрузки (выгрузки) определения в(из) БД.
2) Text (файл Meaning.cs, строка 7). Текст определения. Используется:
   а) button5_Click в *Form1*. Печать в PDF.
   б) *toString*. Является возвращаемым значением.
   в) *CreateMeaningAndSource в Progam*. Добавление нового определения.
   г) *FindRepetitions в Progam*. Поиск повторяющихся определений.
   д) *AddTerm в Source*. Добавление определения для термина.
   е) *AddSource в Term*. Добавление нового определения.
3) *Location* (файл Meaning.cs, строка 7). Документ, в котором дано определение. Используется:
   а) *button5_Click* в *Form1*. Печать в PDF.
   б) *CreateMeaningAndSource в Progam*. Добавление нового определения.
   в) *AddTerm в Source*. Добавление определения для термина.
   г) *AddSource в Term*. Добавление нового определения.
4) *Term* (файл Meaning.cs, строка 8). Термин, к которому относится определение. Используется:
   а) *button5_Click* в Form1. Печать в PDF.
   б) *JoinTerm* добавляет определение к термину, если его нет.
   в) *ChengeRelation*. Удаление определения из термина.
   г) *TurnOff* в *Source*. Если у термина одно определение, то он помечается, как неактуальный.
5) *Source* (файл Meaning.cs, строка 9). Источник определения. Используется:
   а) *JoinSource*. Термин добавляется к источнику.
   б) *ChengeRelation*. Удаление определения из источника.

Ниже описаны поля класса *Source*.

1) *Id* (файл Source.cs, строка 6). ID источника. Использование:
   а) *button1_Click* в *Form5*. Добавление нового источника.
2) *Name* (файл Source.cs, строка 7). Название источника. Использование:
   а) *button5_Click* в *Form1*. Печать в PDF.
   б) *button1_Click* в *Form5*. Добавление нового источника.
   в) *CreateSource* в *Progam*. Добавление нового источника.
   г) *CreateMeaningAndSource* в *Progam*. Добавление нового источника.
   д) *FindRepetitions в Progam*. Поиск повторяющихся источников.
3) *Active* (файл Source.cs, строка 8). Актуальность источника. Использование:
   а) *CreateMeaningAndSource* в *Progam*. Добавление нового источника.
   б) *CreateMeaningAndSource* в *Progam*. Добавление нового источника.
   в) *TurnOff*. Пометка источника неактуальным.
   г) *TurnOn*. Пометка источника актуальным.
4) *Category* (файл Source.cs, строка 8). Категория источника. Использование:
   а) *ChangeCategory*. Смена категории.
5) *Meanings* (файл Source.cs, строка 8). Список определений из источника. Использование:
   а) *button5_Click* в *Form1*. Печать в PDF.
   б) *JoinSource* из *Meaning*. Добавление определения в источник.
   в) *ChengeRelation* из *Meaning*. Удаление определения из источника.
   г) *CreateMeaningAndSource* в *Progam*. Добавление нового источника.
   д) Конструктор *Source*. Инициализация.
   е) *TurnOff*. Пометка определений источника неактуальными.

Ниже описаны поля класса *Term*.

1) *Id* (файл Term.cs, строка 6). ID термина. Использование:
   а) *button1_Click* в *Form3*. Добавление нового термина.

2) *Name* (файл Term.cs, строка 7). Название термина. Использование:
   а) *button5_Click* в *Form1*. Печать в PDF.
   б) *button1_Click* в *Form3*. Добавление нового термина.
   в) *CreateTerm* в *Program*. Добавление нового термина.
3) *EngName* (файл Term.cs, строка 8). Не соответствует документации. Использование:
   а) *CreateTerm* в *Program*. Добавление нового термина.
4) *Abbrev* (файл Term.cs, строка 9). Не соответствует документации. Использование:
   а) *CreateTerm* в *Program*. Добавление нового термина.
5) *Active* (файл Term.cs, строка 10). Актуальность термина. Использование:
   а) *TurnOff* в *Source*. Если у термина одно определение, то он помечается неактуальным.
   б) *TurnOff*. Пометка термина неактуальным.
   в) *TurnOn*. Пометка термина актуальным.
6) *Meanings* (файл Term.cs, строка 12). Определения термина. Использование:
   а) *JoinTerm* в *Meaning*. Добавление определения к термину.
   б) *ChengeRelation* в *Meaning*. Удаление определения из термина.
   в) *TurnOff* в *Source*. Если у термина одно определение, то он помечается неактуальным.
   г) Конструктор *Term*. Инициализация.

DbFile (файл Program.cs, строка 17) является глобальной переменной, хранящей название файла базы данных (*"firstProgram.db"*). Использование:

1) *ResetDatabase*. Сброс базы данных.
2) *CreateSessionFactory*. Копирование базы данных перед началом работы. Получение данных из базы данных
3) *BuildSchema*. Проверка существования базы данных.

BackupFile (файл Program.cs, строка 18) хранит название файла резервной копии базы данных ("BackupFile" + "_" + DateTime.Now.TimeOfDay.ToString().Split(".")[0].Replace(":", "_") + ".db" — инициализируется текущей датой). Используется:

1) *button4_Click* в *Form1*. Восстановление базы данных из резервной копии.
2) *Main*. Удаление резервной копии по окончании работы программы.
3) *ResetDatabase*. Восстановление базы данных из резервной копии.
4) *CreateSessionFactory*. Копирование базы данных перед началом работы.

Глобальный флаг *restartTrigger* (файл Program.cs, строка 19) устанавливается тогда, когда нужно перезапустить приложение для восстановления БД из резервной копии. Использование:

1) *button4_Click* в *Form1*. Устанавливается.
2) *ResetDatabase*. Если флаг установлен, то он сбрасывается и приложение перезапускается.

4 СПИСОК ИЗБЫТОЧНЫХ ФАЙЛОВ И ДИРЕКТОРИЙ

Список избыточных файлов в директории с исходным кодом представлен в Таблице 4.1.

Таблица 4.1 - Список избыточных файлов в директории с исходным кодом.

Файл/Директория	Проблема
Form2.cs, Form3.cs, Form4.cs, Form5.cs	Реализуют недекларированнй функционал
koval_terms.json	Источник данных, который по документации не должен загружаться (база статична).
surprise.dll, surprise.resources	Подозрительные бинарные файлы с неочевидным назначением.
RevDll.dll.base, RevDll.dll.enc	Шифрованные/непрозрачные библиотеки.
appCov.csproj.user	Пользовательские настройки IDE (не нужны в репозитории)
*.cache	Кэш сборки не нужен в репозитории
bin/	Собранные версии приложения не нужны в директории с исходным кодом, так как уже есть в отдельной директории.
obj/	Временные файлы MSBuild

5 БЛОК-СХЕМА ПРОГРАММЫ

Блок-схема программы, созданная с помощью утилиты Understand, представлена на Рисунке 5.1.

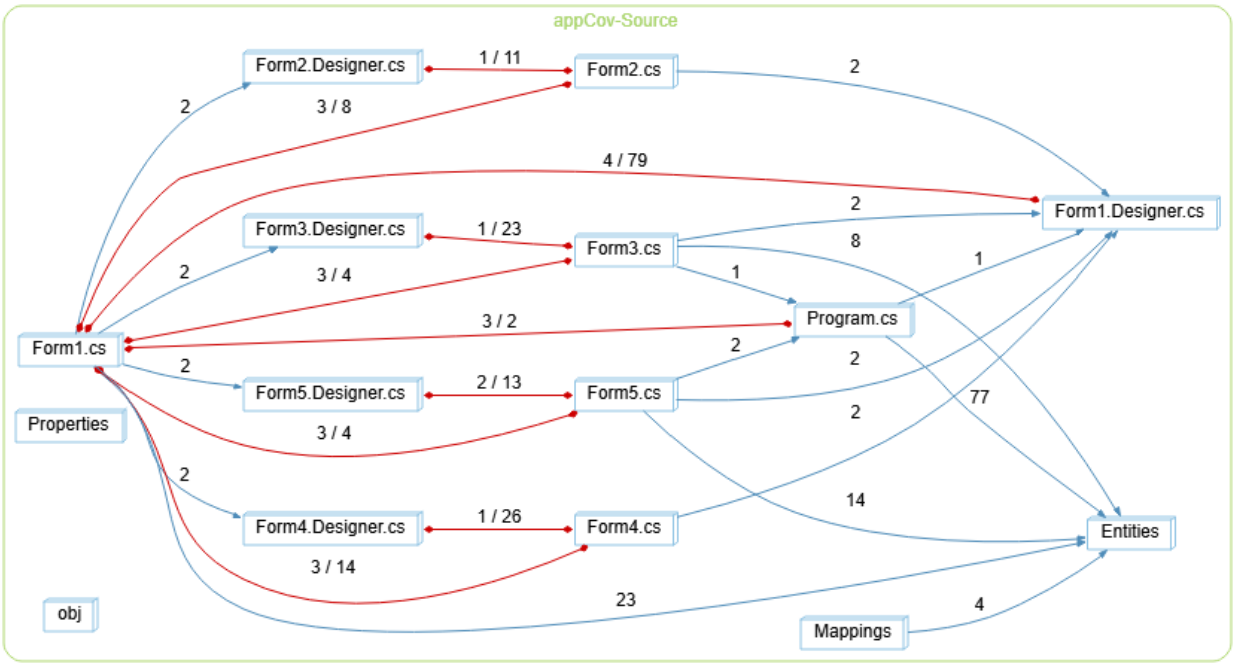

Рисунок 5.1 - Блок-схема программы.

6 СПИСОК ИНСТРУМЕНТОВ, ИСПОЛЬЗУЕМЫХ ПРИ АНАЛИЗЕ

Oracle Virtual Box. Версия 7.1.6 r167084. Для создания изолированной среды для взаимодействия с кодом, предположительно содержащим вредоносный функционал.

Understand 6.5 (Build 1175). Построение графов вызова функций. Построение списка вызывающих функций для глобальных переменных.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы был произведён аудит исходных кодов программы, реализующей функционал просмотра терминов, содержащихся в российских стандартах.

В результате были получены следующие результаты: определены избыточные функции и файлы, построено дерево вызовов программы, определены все глобальные переменные и установлено, в каких фрагментах программы они используются, определены функции, которые не соответствуют документации или выполняют подозрительные действия и проведён их анализ.

Удалось установить, что в исходном коде расположена программная закладка — функция *Surpise,* которая дешифрует динамическую библиотеку неизвестной природы и запускает её. Закладка активируется тогда, когда пользователь выбирает узел, который не имеет родителя (категорию или термин).

Кроме этого, было установлено, что программа имеет множество недекларированного функционала: добавление, изменение и удаление терминов, источников и категорий (хотя документация предполагает только чтение базы данных), наличие отдельной структуры категорий (хотя документация предполагает только деление источников на актуальные и неактуальные), наличие возможности считывать данные из json файла (хотя программа не предполагает наличие входных данных у программы).

Результаты, полученные в рамках данной лабораторной работы, послужат основой для более глубокого анализа проекта в дальнейшем (декомпиляция и анализ бинарных файлов как самой программы, так и обнаруженных подозрительных динамических библиотек).